Contents

```
function msh = load_gmsh4('cube.msh', [])
```

Reads a mesh in msh format, version 1 or 2

```
% Usage:
% To define all variables m.LINES, M.TRIANGLES, etc
% (Warning: This creates a very large structure. Do you really need it?)
%            m = load_gmsh4('a.msh')
%
% To define only certain variables (for example TETS and HEXS)
%            m = load_gmsh4('a.msh', [ 5 6])
%
% To define no variables (i.e. if you prefer to use m.ELE_INFOS(i,2))
%            m = load_gmsh4('a.msh', -1)
%            m = load_gmsh4('a.msh', [])
%
% Copyright (C) 2007  JP Moitinho de Almeida (moitinho@civil.ist.utl.pt)
% and  R Lorphevre(r(point)lorphevre(at)ulg(point)ac(point)be)
%
% based on load_gmsh.m supplied with gmsh-2.0

% Structure msh always has the following elements:
%
% msh.MIN, msh.MAX - Bounding box
% msh.nbNod - Number of nodes
% msh.nbElm - Total number of elements
% msh.nbType(i) - Number of elements of type i (i in 1:19)
% msh.POS(i,j) - j'th coordinate of node i (j in 1:3, i in 1:msh.nbNod)
% msh.ELE_INFOS(i,1) - id of element i (i in 1:msh.nbElm)
% msh.ELE_INFOS(i,2) - type of element i
% msh.ELE_INFOS(i,3) - number of tags for element i
% msh.ELE_INFOS(i,4) - Dimension (0D, 1D, 2D or 3D) of element i
% msh.ELE_TAGS(i,j) - Tags of element i (j in 1:msh.ELE_INFOS(i,3))
% msh.ELE_NODES(i,j) - Nodes of element i (j in 1:k, with
%                       k = msh.NODES_PER_TYPE_OF_ELEMENT(msh.ELE_INFOS(i,2)))
%
% These elements are created if requested:
%
% msh.nbLines - number of 2 node lines
% msh.LINES(i,1:2) - nodes of line i
% msh.LINES(i,3) - tag (WHICH ?????) of line i
%
% msh.nbTriangles - number of 2 node triangles
% msh.TRIANGLES(i,1:3) - nodes of triangle i
% msh.TRIANGLES(i,4) - tag (WHICH ?????) of triangle i
%
% Etc

% These definitions need to be updated when new elemens types are added to gmsh
%
% msh.Types{i}{1} Number of an element of type i
% msh.Types{i}{2} Dimension (0D, 1D, 2D or 3D) of element of type i
% msh.Types{i}{3} Name to add to the structure with elements of type i
% msh.Types{i}{4} Name to add to the structure with the number of elements of type i
%

nargchk(1, 2, nargin);

msh.Types = { ...
    { 2, 1, 'LINES', 'nbLines'}, ... % 1
    { 3,  2, 'TRIANGLES', 'nbTriangles'}, ...
    { 4,  2, 'QUADS', 'nbQuads'}, ...
    { 4,  3, 'TETS', 'nbTets'}, ...
    { 8,  3, 'HEXAS', 'nbHexas'}, ... %5
    { 6,  3, 'PRISMS', 'nbPrisms'}, ...
    { 5,  3, 'PYRAMIDS', 'nbPyramids'}, ...
    { 3,  1, 'LINES3', 'nbLines3'}, ...
    { 6,  2, 'TRIANGLES6', 'nbTriangles6'}, ...
    { 9,  2, 'QUADS9', 'nbQuads9'}, ... % 10
    { 10,  3, 'TETS10', 'nbTets10'}, ...
    { 27,  3, 'HEXAS27', 'nbHexas27'}, ...
    { 18,  3, 'PRISMS18', 'nbPrisms18'}, ...
    { 14,  3, 'PYRAMIDS14', 'nbPyramids14'}, ...
    { 1,  0, 'POINTS', 'nbPoints'}, ... % 15
    { 8,  3, 'QUADS8', 'nbQuads8'}, ...
    { 20,  3, 'HEXAS20', 'nbHexas20'}, ...
    { 15,  3, 'PRISMS15', 'nbPrisms15'}, ...
    { 13,  3, 'PYRAMIDS13', 'nbPyramids13'}, ...
};

ntypes = length(msh.Types);

if (nargin==1)
    which = 1:ntypes;
else
    if isscalar(which) && which == -1
        which = [];
    end
end

% Could check that "which" is properlly defined....

fid = fopen(filename, 'r');
fileformat = 0; % Assume wrong file

tline = fgetl(fid);
if (feof(fid))
    disp (sprintf('Empty file: %s',  filename));
    msh = [];
    return;
end
```

Read mesh type

```
if (strcmp(tline, '$NOD'))
    fileformat = 1;
elseif (strcmp(tline, '$MeshFormat'))
    fileformat = 2;
    tline = fgetl(fid);
    if (feof(fid))
        disp (sprintf('Syntax error (no version) in: %s',  filename));
        fileformat = 0;
    end
    [ form ] = sscanf( tline, '%f %d %d');
    if ((form(1) ~= 2.2)&(form(1)~=2.1))
        disp (sprintf('Unknown mesh format: %s', tline));
        fileformat = 0;
    end
    if (form(2) ~= 0)
        disp ('Sorry, this program can only read ASCII format');
        fileformat = 0;
    end
    fgetl(fid);    % this should be $EndMeshFormat
    if (feof(fid))
        disp (sprintf('Syntax error (no $EndMeshFormat) in: %s',  filename));
        fileformat = 0;
    end
    tline = fgetl(fid);    % this should be $Nodes
    if (feof(fid))
        disp (sprintf('Syntax error (no $Nodes) in: %s',  filename));
        fileformat = 0;
    end
end

if (~fileformat)
    msh = [];
    return
end
```

Read nodes

```
if strcmp(tline, '$NOD') || strcmp(tline, '$Nodes')
    msh.nbNod = fscanf(fid, '%d', 1);
    aux = fscanf(fid, '%g', [4 msh.nbNod]);
    msh.POS = aux(2:4,:)';
    numids = max(aux(1,:));
    IDS = zeros(1, numids);
    IDS(aux(1,:)) = 1:msh.nbNod; % This may not be an identity
    msh.MAX = max(msh.POS);
    msh.MIN = min(msh.POS);
    fgetl(fid); % End previous line
    fgetl(fid); % Has to be $ENDNOD $EndNodes
else
    disp (sprintf('Syntax error (no $Nodes/$NOD) in: %s',  filename));
    fileformat = 0;
end
```

Read elements

```
tline = fgetl(fid);
if strcmp(tline,'$ELM') || strcmp(tline, '$Elements')
    msh.nbElm = fscanf(fid, '%d', 1);
    % read all info about elements into aux (it is faster!)
    aux = fscanf(fid, '%g', inf);
    start = 1;
    msh.ELE_INFOS = zeros(msh.nbElm, 4); % 1 - id, 2 - type, 3 - ntags, 4 - Dimension
    msh.ELE_NODES = zeros(msh.nbElm,6); % i - Element, j - ElNodes
    if (fileformat == 1)
        ntags = 2;
    else
        ntags = 3; % just a prediction
    end
    msh.ELE_TAGS = zeros(msh.nbElm, ntags); % format 1: 1 - physical number, 2 - geometrical number
                                            % format 2: 1 - physical number, 2 - geometrical number, 3 - mesh partition number
    msh.nbType = zeros(ntypes,1);
    for I = 1:msh.nbElm
        if (fileformat == 2)
            finnish = start + 2;
            msh.ELE_INFOS(I, 1:3) = aux(start:finnish);
            ntags = aux(finnish);
            start = finnish + 1;
            finnish = start + ntags -1;
            msh.ELE_TAGS(I, 1:ntags) = aux(start:finnish);
            start = finnish + 1;
        else
            finnish = start + 1;
            msh.ELE_INFOS(I, 1:2) = aux(start:finnish);
            start = finnish + 1; % the third element is nnodes, which we get from the type
            msh.ELE_INFOS(I, 3) = 2;
            finnish = start + 1;
            msh.ELE_TAGS(I, 1:2) = aux(start:finnish);
            start = finnish + 2;
        end
        type = msh.ELE_INFOS(I, 2);
        msh.nbType(type) = msh.nbType(type) + 1;
        msh.ELE_INFOS(I, 4) = msh.Types{type}{2};
        nnodes = msh.Types{type}{1};
        finnish = start + nnodes - 1;
        msh.ELE_NODES(I, 1:nnodes) = IDS(aux(start:finnish));
        start=finnish + 1;
    end
    fgetl(fid); % Has to be $ENDELM or $EndElements
else
    disp (sprintf('Syntax error (no $Elements/$ELM) in: %s',  filename));
    fileformat = 0;
end

if (fileformat == 0)
    msh = [];
end
```

```
fclose(fid);
```

This is used to create the explicit lists for types of elements

```
for i = which
    if (~isempty(msh.Types{i}{3}))
        cmd = sprintf('msh.%s=msh.nbType(%d);', msh.Types{i}{4}, i);
        eval(cmd);
        % Dimension
        cmd = sprintf('msh.%s=zeros(%d,%d);', msh.Types{i}{3}, msh.nbType(i), msh.Types{i}{1}+
1);
        eval(cmd);
        % Clear nbType for counting, next loop will recompute it
        msh.nbType(i) = 0;
    end
end

for i = 1:msh.nbElm
    type = msh.ELE_INFOS(i,2);
    if (find(which == type))
        if (~isempty(msh.Types{type}{3}))
            msh.nbType(type) = msh.nbType(type)+1;
            aux=[ msh.ELE_NODES(i,1:msh.Types{type}{1}), msh.ELE_TAGS(i,1) ];
            cmd = sprintf('msh.%s(%d,:)=aux;', msh.Types{type}{3}, msh.nbType(type));
            eval(cmd);
        end
    end
end

return;
```

```
Error using dbstatus
Error: File: /Users/Almire/Documents/MATLAB/load_gmsh4.m Line: 1 Column: 27
Invalid expression. Check for missing multiplication operator, missing or unbalanced delimiter
s, or other syntax error. To construct matrices, use brackets instead of parentheses.
```

Published with MATLAB® R2018a