FEM Implementation within MATLAB

Starting this chapter, where we describe the realization of DAEdalon within the MATLAB environment, we quickly itemize the main features of MATLAB operating on vector and matrix representations, respectively. This list can not substitute a deeper study of the printed MATLAB manuals or any tutorial obtained from the internet or even the built-in help function. This fact is recognized very quickly: MATLAB can not be learned by reading, but only by programming in the environment!

4.1 MATLAB Basics

- A “vector” in the physical (Euklidian) space $\mathbb{E}^3$ is read as column array of the three coefficients $a_1, a_2$ and $a_3$ in the form

$$\mathbf{a} := \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \quad (4.1)$$

represented and implemented in MATLAB by $\mathbf{a} = [a_1; a_2; a_3]$, see (2.1). With respect to the continuum mechanical background in Chap. 2, we avoid here a further notation of the tensorial basis system and denote all vectors and tensors in the above given (Euklidian) space, if nothing else is mentioned.

- In contrast to that, a transposed vector $\mathbf{a}^T$ is given by $\mathbf{a}'$ or directly by $\mathbf{a}_{\text{tr}} = [a_1 \ a_2 \ a_3]$ as row type array. Please note, the semicolon “;” as given above forcing a line break are omitted here in order to represent a row vector.

- Consequently, the vector (dot) product, see (2.2), ending in a scalar quantity is given by $\mathbf{c} = \mathbf{a}_{\text{tr}} \ast \mathbf{a}$ or $\mathbf{c} = \mathbf{a}' \ast \mathbf{a}$.

- Further on, a $3 \times 3$ array (dyadic) representation is given e.g. by $\mathbf{C} = \mathbf{a} \ast \mathbf{a}'$, see (2.3).

- MATLAB coding is case sensitive, so that $\mathbf{c}$ is not equal to $\mathbf{C}$!
• In addition, we often make use of the command
  \[
  \text{reshape}(C,[\text{num\_col} \text{ num\_row}]),
  \]
  where \text{num\_col} represents the new number of columns and \text{num\_row} the
  new number of rows of the array \text{C}. Please note, that the total number of
  entries \text{num\_col}\times\text{num\_row} remains constant during this operation.
• Finally, we want to emphasise the “blockwise” operation and storing tech-
  nique of MATLAB, where one can operate and manipulate on single parts
  of a (large) array. We make use of that functionality especially during
  the assembling procedure, see Sec. 4.4.1, or directly in \text{stiffness\_func\_m},
  which contains the command line
  \[
  k(\text{pos\_vec},\text{pos\_vec}) = k(\text{pos\_vec},\text{pos\_vec}) + k_{\text{elem}};
  \]
  adding directly the element contributions into the global system array.

### 4.2 Structure of DAEdalon

Since the first steps in the development of DAEdalon, in the following of the
year 2000, many contributions and details are added — and are ongoing
added actually, so that the aim of the whole booklet is a representation of
the main and basic structure and the architecture of the system behind.

Principally, the main skeleton of DAEdalon is given schematically in Fig. 4.1,
which represents the m\text{-file} structure of the system. The initialization of
the program and loading of the actual problem is activated by \text{dae} and \text{lprob}, re-
spectively. The time counter is set by the \text{time command}. At the that state, the
capabilities described in Sec. 4.5 can be obtained to check or control the defined

\begin{itemize}
  \item \text{dae.m}
  \item \text{lprob.m}
  \item \text{time.m}
  \item \text{go.m} \rightarrow \text{stiffness.m}
  \item \text{syst.m}
  \item \text{solv.m}
  \item \text{residuum.m}
\end{itemize}

\begin{figure}[h]
\centering
\begin{tikzpicture}
\node (daem) at (0,0) {dae.m};
\node (lprobm) at (1,-1) {lprob.m};
\node (timem) at (2,-2) {time.m};
\node (gom) at (3,-3) {go.m};
\node (stiffnessm) at (4,-4) {stiffness.m};
\node (systm) at (4,-5) {syst.m};
\node (solvem) at (4,-6) {solv.m};
\node (residuumm) at (4,-7) {residuum.m};
\draw[->] (gom) -- (stiffnessm);
\end{tikzpicture}
\caption{Schematic Carry–Out of a DAEdalon Analysis}
\end{figure}
problem. The numerical solution is activated afterwards by go, which is just an assembly of the in Fig. 4.1 visualized commands. Obviously, that is the core part of DAEdalon representing a development tool for FE.

Please, feel free to add your own developments and ideas within that given structure ...

4.3 Preprocessing

DAEdalon requests at least for the six input data files node.inp, el.inp, mat1.inp, geom.inp, displ.inp and force.inp as already explained in Sec. 1.5. As mentioned before, the focus of DAEdalon is the solution part of the FEM, for which we decide to neglect sophisticated preprocessing tools available in many CAD– and special preprocessing software–packages in meantime.

4.3.1 Structure of Input-Files and Processing

geom.inp
    line 3 — nummat
    line 5 — ndf
node.inp
    numnp
    ndm
el.inp
    numel
    nel

After reading that basic input data files, the system computes and initializes automatically the global variables and array dimensions, e. g. total number of degrees of freedom gesdof = ndf*numnp giving the size of the solution vector u and its increment du, the size of the right hand side rhs.

4.4 Numerical Solution

The main task of the numerical implementation is the solution of (2.52) with respect to the solution vector u, which can be realized in DAEdalon along with Fig. 4.1.

4.4.1 Assembly Procedure — The Global System

The two core commands on the global level of FE analysis within DAEdalon are the scripts stiffness.m and syst.m, which assemble the global system matrix and the right hand side vector and consider, finally, for the given boundary conditions.
In detail, the `stiffness.m` script loops over all given elements in all given material sets and collect the element information, i.e. the element stiffness `kmat` given in (2.53) and the element right hand side vector `rhs`, see Sec. 2.55. For each element call, the necessary information as actual displacements `ue` or the connected history data is selectively given out from the global fields.

The following is the assembly procedure of the global system, which is in that way the main advantage of doing it within the MATLAB environment. We are able to handle the particular element information as sub-/block-matrix and store it very easily in the global arrays by operations like

```
  kmat(1:2, 3:4) = kmat_elem()
```

resulting in the global system matrix `K`.

Further on, `syst.m` considers for the applied (displacement) boundary conditions and the load array. Here, each given (displacement) boundary condition leads to a reorganization of the global system of equations. Because of its known entry, the respecting information is sorted into the right hand side of the system and the corresponding column and row of `kmat` is filled completely with zero entries, except for the diagonal entry, which is filled with “1”. So, the global system remains it original size `nump * ndf` equations, but the system matrix `kmat` gets not singular and is invertible by that treatment.

### 4.4.2 Newton–Iteration

As described in Sec. 2.7.2, we start with (2.52) and the assembled global stiffness resulting from (2.53). Numerically, we solve (2.55), which is given as

```
  kmat * du = rhs ,
```

by

```
  du = kmat \ rhs .
```

Here, `rhs` represents the discretization of $-g(\hat{u})$ in vector form. Doing so, we are able to update `u` by

```
  u = u + du
```

within the iteration loop, see `solv.m`. In these loops we control the quality of the solution by the norm

```
  res_norm=sqrt(rhs'*rhs)
```

of resulting right hand side vector `rhs`, see `residuum.m`. The user is free to end the iteration process, if that norm falls below a tolerance of e.g. $10^{-9}$, which denotes numerically the zero.
4.5 Postprocessing

4.5.1 Mesh Representation, Loads and Boundary Conditions

We provide a number of predefined m-files for the postprocessing and the visualization of the obtained FE results. Of special interest are the commands mesh0, meshx, nodenum and elnum for the representation of the mesh state in the undeformed and deformed configuration and the numbering of the nodes and elements, respectively.

In addition, the forc and boun commands give the applied nodal loads as arrows and the displacement restraints as line at the subjected nodes directed in the given direction. Both commands represent the content of the input data base given in forc.inp and displ.inp.

4.5.2 Contour-Plots

The treatment and the computation of a requested quantity by the cont(·) command is explained in the following. To compute a smooth field \( p \) in the sense of a least square fit from e.g. the (discrete) scalar pressure \( p^h \) projected to the FE nodes obtaining the nodal values \( p_{\text{node}} = \sum I N_I p_I^{\text{node}} = N^T \cdot p_{\text{node}} \), see also (2.39), we consider the minimization

\[
\int_{\mathcal{B}} \frac{1}{2} (p - p^h)^2 dv \to \min \quad \int_{\mathcal{B}} \delta p (p - p^h) dv = 0 . \tag{4.2}
\]

These quantities can be prepared elementwise and stored there as

\[
\mathbf{A} \int_{\mathcal{B}} \mathbf{N} \cdot \mathbf{N}^T dv \mathbf{p}_{\text{node}} = \mathbf{A} \int_{\mathcal{B}} \mathbf{N} \cdot p^h dv , \tag{4.3}
\]

which can be solved very easy for \( \mathbf{p}_{\text{node}} \). So, the nodal contribution \( \mathbf{p}_{\text{node}} \) from each element is obtained by a assembly procedure over all elements very analogous to the construction of the element stiffness matrix and the load vector. For that reason, we compute the numerator and the nominator contribution out of (4.3) in the same loop within the element formulation and store them in the arrays cont.zaehtler and cont.nener, respectively.

4.5.3 GUI — Realization of a Graphical Environment

The above given commands — and some additional features of DAEdalon — can also be obtained and executed by a Graphical User Interface activated
by the DAE Control Button in the plot window as is given exemplary in Fig. 4.2. Nevertheless, again the user is able to add his own ideas and to modify that given version of the GUI.

### 4.6 Writing Extensions for DAEdalon

For a demonstration of the coding of a finite element formulation on principle, we refer to `elem4.m`, which has already been considered in Sec. 1.2 for a first introduction and show a formulation considering for rotational symmetry and the EAS formulation of that element enabling us to model highly incompressible material behavior. In addition, we describe the interface for implementing material models and give the formulation of a general purpose 2- or 3-d truss element with nodes.

#### 4.6.1 Continuum Element `elem4.m` with 4-Nodes in Plane Strain

The implementation of a finite element within DAEdalon is given as MATLAB function in the form
function [k_elem, r_elem, cont_zaeher, cont_nenner, ...  
    hist_new_elem, hist_user_elem] = ...  
    elem4(isw, nel, ndf, contvar, mat_name, mat_par, x,u_elem,  
    hist_old_elem, hist_user_elem)

where the nodal positions x and the nodal displacements u_elem, the material  
parameters mat_par and the history data are given besides the number of  
nodes nel and the number of degrees of freedom per node ndf.

Generally, one has to define the element stiffness in form of the matrix

\[ K_{elm} := K_{mat} + K_{geom}, \]

(4.4)

see (2.53), and the element residual or load vector

\[ r_{elm} := \int B^T \cdot \sigma \, dv, \]

(4.5)

see (2.52), respectively. In Sec. 2.6.2 we have seen a description of the B–  
matrix.

This is realized e.g. in the above mentioned coding of elem4.m as

for aktgp=1:umgp
    ...
    k_elem = k_elem + b' * D_mat * b * dv;
    r_elem = r_elem + b' * sig * dv;
    ...
end % Loop aktgp

as the loop over the integration point representing the volume integration as  
shown in Sec. 2.6.4. In addition, that loop contains the computation as well  
of the resulting stress vector \( \sigma \) including the material modulus \( D \) in the form  
of \( D_{mat} \) as the \( B \)-matrix as array of the derivatives of the shape functions.

The cont variables are used for the contour plot functions and the mapping  
of the data therein and define the nominator and the numerator in (4.3).

4.6.2 Formulation for Rotational Symmetry of  
extem4.m

The above presented prototype element formulation of a 4–noded continuum  
element can also be used for axisymmetric descriptions of rotational bodies  
as given in Fig. 4.3. Just small modifications are required in order to describe  
an axisymmetric 3–dimensional body under rotational load and deformation  
conditions. For that, we apply the following cylindrical coordinate system  
with the labels \( x_1 \) for the radial, \( x_2 \) for the axial direction and \( x_3 \) for the  
tangential or circumferential direction, respectively. As given in that formulation,  
the axial direction has always to be adjusted to \( x_2 \) as rotational axis.
The given element formulation is based to the deformation gradient for that assumptions as

\[
F = \begin{bmatrix}
1 + \frac{\partial u_r}{\partial x_1} & \frac{\partial u_r}{\partial x_2} & 0 \\
\frac{\partial u_2}{\partial x_1} & 1 + \frac{\partial u_2}{\partial x_2} & 0 \\
0 & 0 & r/R
\end{bmatrix},
\] (4.6)

which is the basis of the strain computation in the constitutive model.

Following this notation, we obtain the stress response in the same order, such that the first index gives the radial component and the second one the axial component. As in the plane strain element formulation in Sec. 4.6.1, we store the shear response \(\sigma_{12}\) in the third component. In Sec. 5.2 we give an application of that formulation.

### 4.6.3 EAS Expansion of elem4.m

In order to overcome some disadvantages of the presented prototype element formulation given in `elem4.m`, many modifications of that element are well known in literature, see e.g. Belytschko et al. [1984] for a first proposal. The main disadvantage of the element is given by its too stiff response ("locking behavior"), especially in mechanical cases which are dominated by bending or whose material behavior is going to the limit case of incompressibility.
4.6 Writing Extensions for DAEdalon

The simplest and most intuitive idea was to modify the integration scheme of the element especially in the volumetric part of the stress response from a 2 × 2 version as given in Sec. 2.6.4 into an 1–point scheme, which soften the element behavior. Nevertheless, this modification raises in an additional lack of that formulation: In unprofitable loading cases the element may response with physically not motivated deformation modes, which are denoted due to the resulting shapes as hourglassing.

Today, the most attractive and modern version of element formulations is known as enhanced assumed strain (EAS) formulation, which are based mainly on SIMO & ARNERO [1992] and ANDELFINGER & RAMM [1993].

This section should describe the main underlying idea of that concept, which is widely used in many element formulation, generally to prevent too stiff element responses. Starting point of such formulations is an enhancement of the variational principle 2.33 into a 3–field functional (Hu–WASHIZU principle) with an internal energy expression

$$\Pi^{e,int} := \int_{V_e} \frac{1}{2} \varepsilon_h^T \cdot C \cdot \varepsilon_h - \sigma_h^T \cdot \varepsilon_h + \sigma_h^T \cdot B \cdot \mathbf{u}_h \, \mathrm{d}V_e \tag{4.7}$$

introducing the enhanced strains $\tilde{\varepsilon}_h$ in addition to the (original) compatible strains in

$$\varepsilon_h = \underbrace{B \cdot \mathbf{u}_h}_{\text{compatible}} + \underbrace{\tilde{\varepsilon}_h}_{\text{enhanced strains}} \tag{4.8}$$

Obviously, this additional and fictive strain field may not cause any energy turnover within the element formulation, which delivers the additional condition

$$\int_{V_e} \sigma_h^T \cdot \tilde{\varepsilon}_h \, \mathrm{d}V_e = 0 \tag{4.9}$$

in an elementwise view. In that sense, the introduced strain field $\tilde{\varepsilon}_h$ may be incompatible to the mechanical strains and do not give any amount to (4.7).

Furthermore, the FE interpolation of

$$\varepsilon_h = \sum_k M_k \alpha_k = \mathbf{M} \cdot \mathbf{\alpha} \tag{4.10}$$

in analogy to (1.7) is given by a matrix representation $\mathbf{M}$ with $k$ columns, which is discussed below.

Following the argumentation as in the standard element formulation, leads to a system of equations on the element level of the form

$$\begin{bmatrix} \mathbf{D} & \mathbf{L} \\ \mathbf{L}^T & \mathbf{K}^e \end{bmatrix} \cdot \begin{bmatrix} \mathbf{\alpha} \\ \mathbf{u}^e \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{l}^e \end{bmatrix} \tag{4.11}$$
where \( \alpha \) denotes here additional degrees of freedom and the block matrices are given by

\[
K^e = \int_{V_e} B^T \cdot C \cdot B \, dV_e \tag{4.12}
\]

as before, see (1.12), and

\[
D = \int_{V_e} M^T \cdot C \cdot M \, dV_e ,
\]

\[
L = \int_{V_e} M^T \cdot C \cdot B \, dV_e . \tag{4.13}
\]

Assuming to know all expressions in (4.11), one obtains a modified element stiffness matrix

\[
K^{e, \text{mod}} = K^e - L^T \cdot D^{-1} \cdot L
\]

by a local condensation of the system on the element level, which is assembled to the global system as described before.

To complete this short introduction into the EAS element formulation, we show the determination of the matrix \( M \) in (4.10), which is described and motivated in detail in Andelfinger & Ramm [1993] by a discussion on the deficiencies of the classical element shape functions given in (2.38) with respect to a complete set of polynomial terms. It is shown, that the best approximations can be obtained by setting

\[
M := \frac{\det J_0}{\det J} T_0^{-T} \cdot M_\xi , \tag{4.15}
\]

where the index 0 denotes the evaluation of \( J \), see (2.43), and

\[
T_0 = \begin{bmatrix}
J_{011}^2 & J_{012}^2 & 2J_{011} J_{012} \\
J_{021}^2 & J_{022}^2 & 2J_{021} J_{022} \\
J_{011} J_{021} & J_{012} J_{022} & J_{011} J_{022} + J_{012} J_{021}
\end{bmatrix} \tag{4.16}
\]

in the respective element center at \( \xi = \eta = 0 \).

Finally, the matrix \( M_\xi \) with additional terms of the shape functions is given for \( k = 4 \) by

\[
M_\xi = \begin{bmatrix}
\xi & 0 & 0 & 0 \\
0 & \eta & 0 & 0 \\
0 & 0 & \xi & \eta
\end{bmatrix} , \tag{4.17}
\]

which is shown to be sufficient for the \texttt{elem4} formulation. This formalism can expanded easily for \( k > 4 \) by adding columns to \( M_\xi \) containing additional term as e.g. \( \xi \eta \).

Doing so, a very versatile usable element formulation can be realized, which can be applied in many structural mechanics tasks especially those with a
nearly incompressible material response as metal plasticity or in rubber components.

4.6.4 **General Truss Element** `elem10.m with 2 Nodes`

In order to formulate this simple finite element, the weak form (2.33) sounds like

\[ \int_l A \sigma \delta \varepsilon \, dx - F_1 \delta u|_{\partial B_e} = 0, \quad (4.18) \]

where \( A \) denotes the truss cross section, \( \sigma \) the axial stress and \( F = A \sigma \) the axial (outer) force loading the truss. As given in Fig. 4.4, the axial displacement \( u \) is obtained by the projection of \( t \) and the spatial displacements

\[ u = t \cdot u = \sum_{I=1}^{2} N_I(\hat{x}) t \cdot u_I, \quad (4.19) \]

with the local shape functions

\[ N_1(\hat{x}) = 1 - \frac{\hat{x}}{l_e} \quad \text{and} \quad N_2(\hat{x}) = \frac{\hat{x}}{l_e}. \quad (4.20) \]

With that, we can easily obtain the local strain as

\[ \varepsilon = \frac{d}{d\hat{x}} u = \sum_{I=1}^{2} \frac{dN_I(\hat{x})}{d\hat{x}} t \cdot u_I = \sum_{I=1}^{2} B_I \cdot u_I \quad (4.21) \]

or as given e.g. in (1.7) collecting the derivatives of the element shape functions in a \( B \)-matrix, here given by \( B_I = N_{I,\hat{x}} t^T \). In the following, we give

![Fig. 4.4 2–Noded Truss Element](image-url)
the numerical realization of that derivations in a form as implemented in elem10.m:

```matlab
% material parameters
E_mod = mat_par(2);
A = mat_par(3);

% normalized axial vector and length
t = x(2,:)' - x(1,:)';
L = norm(t);
t = t/L;

% local displacement
u_axial = (u_elem(2,:) - u_elem(1,:))*t;

% strain
epsilon = u_axial/L;

% stress / Hooke-model
sig = E_mod*epsilon;

% B-matrix
B = [-t',t']/L;

% element stiffness k_elem
k_elem = B' * E_mod*A * B * L;

% element residual vector
r_elem = B' * sig*A*L;
```

Again, the main results of that MATLAB (element) function is the element stiffness matrix $K^\text{elmt}$ in $k\_elem$ and the element residual vector $r^\text{elmt}$ in $r\_elem$, respectively.

### 4.6.5 Material Models

Interested in applying and testing own, new material formulations, one can use likewise material interface functions as given exemplary such as mat33.m in the form

```matlab
function [sig,vareps,D_mat,hist_new_gp,hist_user_gp] ...
    = mat33(mat_par,F,hist_old_gp,hist_user_gp)
```

In that sense, please give the strain measure e.g. $E$ or $e$, see (2.17) and (2.18), respectively, in `vareps` in VOIGHT structure as in (2.29). The material model should compute the conjugated stress response $S$ or $\sigma$ in the array `sig` and the material modulus $\mathbf{D}$ in `D_mat`, see Sec. 3.1.2.
Here, we focus on the local NEWTON iteration used in Sec. 3.2.3 for the
determination of plastic strain increments $\Delta \varepsilon^p_{n+1}$:

```plaintext
while abs(phi) > 1.0E-12
    dphi = -2.0*mu - 2.0/3.0 * Emod/N * 
        (Emod/Y0*(alpha+sqrt(2.0/3.0)*gamma)+1.0)^(1.0/N -1.0);
    dgamma = - phi/dphi;
    gamma = gamma + dgamma;
    phi = dev_tau_abs - sqrt(2.0/3.0)*Y0 * (Emod/Y0*(alpha +
        sqrt(2.0/3.0)*gamma)+1.0)^(1.0/N) - 2.0*mu*gamma;
end % Newton-Iteration
```

In the case of e.g. damage models or other highly sophisticated material
descriptions, one should use the same interface function to compute the
increments of the considered internal variables. As mentioned before, in BAASER & TVERGAARD [2003] exemplary the implementation of the ROUSSELIER
damage model is shown, which is treated exactly in that sense.