CraFT user's guide

Hervé Moulinec

December 12, 2014

1 Introduction

In this chapter, one will describe how to run CraFT. That means what sort of input data CraFT needs, how to specify it to CraFT, what sort of output data are to be created, and how these data (input and output) are organized.

To describe the mechanical problem she/he wants to run, the user must:

- describe the geometry of the microstructure via an image telling which phase each pixel belongs to
- describe the mechanical behavior of each phase, this is done in CraFT via two files:
  - a file describing all materials present in the microstructure: the type of behavior they obey (linear elasticity, elastic-perfectly plastic behavior, ...), and their peculiar mechanical properties (e.g. Young's modulus and Poisson coefficient in the cas of isotropic linear elasticity)
  - a file telling for each phase which material it belongs to, and how the material is oriented in that phase
- describe the loading conditions,
- tell what outputs the user wants to store at the end of the computation, and the name she/he wants to give to them
- give some tuning parameters of the method:
  - how to choose the reference material C_0
  - precision required for convergence of the iterative process

2 CraFT Usage

The simplest way to run CraFT is to run it interactively by typing:

```
craft -i
```

(with `-i` "interactive mode" option) or simply by:

```
craft
```

(without any options).

Then, CraFT asks 7 questions to the user:

- the name of an image file describing the microstructure
- the name of a file describing the phases of the microstructure
- the name of a file describing the materials the phases are made of
- the name of a file describing the loading conditions
- the name of a file in which the user specify the outputs she/he wants
- how to choose the "reference material" C_0
- the required accuracy (the accuracy at which iterative processes for convergence have to be stopped)

Each of this specification will be detailed in next sections.

If the user prefers to run CraFT in a single line of command, she/he can user `-f` option followed by the name of a file describing the inputs required by CraFT:

```
craft -f inputfile
```

See section 3.1, page 4 for more details on input file format.

An other possibility, is to specify every input parameters separately by using adequate input options: `-c -p -m -l -o -C -e`

These options must be used together. Invoking them exclude use of `-f` or `-i` options (and vice versa).

Table (2.1) summarizes all possible options of `craft` command.

`-h`	displays help
`-v`	verbose mode (default: non verbose)
`-V`	displays craft version number
`-i`	interactive mode: inputs are asked to the user (it is the default mode)
`-f <file>`	read inputs in file `<file>`
`-c <file>`	microstructure is given in file `<file>`
`-p <file>`	phases are described in file `<file>`
`-m <file>`	materials are described in file `<file>`
`-l <file>`	loading conditions are described in file `<file>`
`-o <file>`	outputs are described in file `<file>`
`-C <line>`	C_0 is specified in command line `<line>`
`-e <accuracy>`	accuracy required in `<accuracy>` (`<accuracy>` can be either a single value or two values separated by a comma for accuracy required for stress divergence and for accuracy required for loading conditions)
`-n <threads>`	number of threads to be used (in case of OpenMP compiled version)

Table 2.1: CraFT options

3 Entering specifications of a given problem

3.1 Input file

The input specifications of a problem can be given to CraFT in a file, the name of which is entered by `-f` option.

Input file can:

- either contain exactly what should have been answered to CraFT when interactively called, in the same order,
- or can use keywords to enter specifications; in that case the different specifications can be entered in any order.

The two formats for entering specs can not be mixed together.

In both cases, a line beginning with a # character is considered as a comment line. An empty line (i.e. a line containing nothing or just white spaces) is ignored.

Format of input file without keywords:

In format without keywords, spec have to be entered in the strictly same order as in interactive mode:

- the name of an image file describing the microstructure
- the name of a file describing the phases of the microstructure
- the name of a file describing the materials the phases are made of
- the name of a file describing the loading conditions
- the name of a file in which the user specify the outputs she/he wants
- how to choose the "reference material" C_0
- the required precision (i.e. the accuracy at which iterative processes for convergence have to be stopped)

An example of an input file without keywords is given in annex A.2, page 20.

Format of input file with keywords:

An input file using keywords contains lines beginning with one of the available keywords (summarized in table 3.1). The keyword is followed by a = character, and then by the specification itself. Case distinction is ignored in keywords. Blanks are ignored.

keywords	arguments
`microstructure`	the name of an image file describing the microstructure
`phases`	the name of a file describing the phases of the microstructure
`materials`	the name of a file describing the materials the phases are made of
`loading`	the name of a file describing the loading conditions
`output`	the name of a file in which the user specify the outputs she/he wants
`C0`	how to choose the "reference material" C_0
`precision`	the required precision

Table 3.1: Keywords available in input files

An example of an input file with keywords is given in annex A.3, page 21.

For keywords accepting a file name as argument (i.e.: `microstructure` ,`phases`, `materials` , `loading` and `output`), it is also possible to directly enter the content of the file into the input file. In that case, the keyword is followed by the content of the file enclosed by braces ({ and }).

Example A.4 in page 22 illustrates this case.

3.2 File describing the microstructure

The microstructure of the problem treated is described by an image file in CraFT format or in "simple legacy" VTK format.

In both format, each pixel of a microstructure image must contain the index of the phase it belongs to.

(see section (4.1), page 15, for more details on digital images).

See `www.vtk.org/VTK/img/file-formats.pdf` for details on simple legacy VTK file format.

3.3 File describing the phases

Phases in the microstructure are described in an ascii file, in which every phase in the microstructure is described by a line. First column gives the number of the phase, second column gives the number of the material the given phase bis composed of, the next three colums give the orientation of the material in the given phase by three Euler angles ϕ_1, Φ, ϕ_2 (see figure 3.2 for details).

Empty lines and lines beginning with # character are ignored by CraFT.

Remarks:

```
#-------------------------------------------------------------------
#  phase  material     phi1           Phi            phi2
#-------------------------------------------------------------------
     0        0       3.8135413      1.8862685      1.1466009
     1        0       2.7503878      1.7827771      4.2127749
     2        0       2.0567105      1.6476569      3.3482159
     3        0       4.3410043      1.1427749       3.907608
     4        0       3.5043039      1.4998321      4.8580132
     5        0       4.4619361      1.6873032      6.1930471
     6        0        4.028708        2.07412      5.1103068
     7        0       2.2259622      1.4106619      1.8815486
     8        0       1.0618022      2.0650686     0.89195132
     9        0       4.6502682      1.5093459       5.475368
```

Figure 3.1: example of file describing the phases of a microstructure. All of the 10 phases are composed of the same material (whose id is 0) but do have different crystalline orientations.

- A given phase is uniquely described by an id number.
- Phases are not necessarily numbered from 0 to n: phase file has just to describe every phase present in image file of the microstructure, however the phases are numbered in the image.
- Phases in phase file can be more numerous that actual phases in microstructure image: the only prescription is that every phase in the microstructure must be described in phase file.

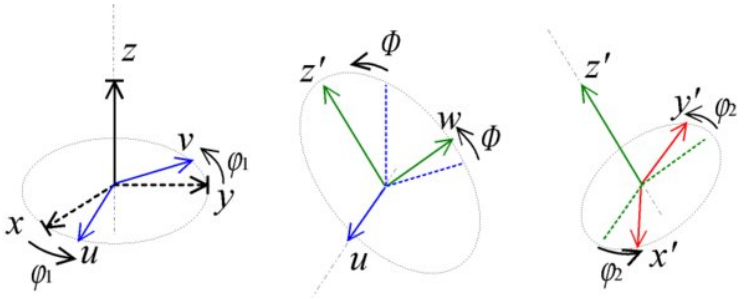


Figure 3.2: Euler angles with Bunge notations

3.4 File describing the materials

A unique file describes all phases of the microstructure. This is an ascii file composed of paragraphs each of one describing a given material.

Each paragraph begins with a line telling the id number of the material and the id number of the constitutive law that the material obeys.

id	constitutive law
0	void
1	isotropic linear elasticity
2	elastic perfectly plastic behavior (with isotropic elasticity)
3	anisotropic linear elasticity

Table 3.2: CraFT identification numbers of constitutive laws

The next lines give the value of the parameters of the constitutive law; their number, types and order depending on the constitutive law and how it has been implemented. For example, for an isotropic linear elastic behavior just two parameters has to be entered: Young's modulus and Poisson coefficient and for an elastic perfecftly plastic behavior, three parameters are required: Young's modulus, Poisson coefficient and yield stress.

Table 3.2 summarizes the ids of the different constitutive laws which has been implemented till now.

Details of parameters to be entered for each behavior are given in appendix (B).

Empty lines are ignored and lines beginning with # character are considered as commentar by CraFT in material description files.

3.5 Loading specifications

Files specifiying loading conditions consist in two parts:

- loading condition (prescribed stress, prescribed strain or prescribed direction of stress)
- one or more lines describing every loading step(s)

3.5.1 loading condition

CraFT enables three different loading conditions:

- prescribed macroscopic strain : macroscopic strain E is imposed
- prescribed macroscopic stress: macroscopic strain $\boldsymbol{\sigma}$ is imposed
- prescribed direction of stress + prescribed strain in that direction: macroscopic stress $\boldsymbol{\sigma}$ has to be colinear to prescribed direction of stress $\boldsymbol{\sigma}_0$ and the product of macroscopic stress and macroscopic strain, i.e. $\boldsymbol{\sigma} : E$, is prescribed.

Loading specification file begins with a line containing a letter:

- `D` : prescribed macroscopic strain
- `C` : prescribed macroscopic stress
- `S` : prescribed direction of stress

3.5.2 Loading steps

Loading steps may be specified step-by-step: a given line describes one given step, or implied loops may be used to specify several steps in one line.

"Step-by-step" specification

A basic line of loading step specification comprises 8 values:

- the time value (for example in seconds), hereafter called t
- the 6 components of a symmetrical 2d order tensor, hereafter called $\mathbf{d}$ supposed to be entered in the following order: 11, 22, 33, 12, 13, 23
- a scalar, hereafter called k

k and $\mathbf{d}$ do have different meaning depending on loading condition:

- in the case of prescribed macroscopic strain : macroscopic strain $\mathbf{E}$ at time t is given by $\mathbf{E} = k.\mathbf{d}$
- in the case of prescribed macroscopic stress: macroscopic stress $\mathbf{\Sigma}$ at time t is given by $\mathbf{\Sigma} = k.\mathbf{d}$
- in the case of prescribed direction of stress, the macroscopic stress must be colinear to $\mathbf{d}$ (in other words: $\mathbf{d}$ is the direction of stress) and the product of the macroscopic strain by $\mathbf{d}$ must be equal to k: $\mathbf{E} : \mathbf{d} = k$

Important note: CraFT implies that at time $t = 0$, loading modulus k is null and direction d is useless.

For example:

```
#-------------------------------------
# prescribed strain
D
#-------------------------------------
# loading
#t      direction             k
#       11 22 33 12 13 23
#- - - - - - - - - - - - - - - - - - -
0.1      1. 0. 0. 0. 0. 0.     2.
#
```

In this file, macroscopic strain is prescribed, the loading consists in one step at time: $t = 0.1s$, the macroscopic strain $\mathbf{E}$ must be equal to $(E_{11}, E_{22}, E_{33}, E_{12}, E_{13}, E_{33}) = (2, 0, 0, 0, 0, 0)$

Implied loop specification

In the case of a monotonic loading, it can be tedious to enter the lines of every required time steps, where the time values t and the loading modulus k change regularly from one step to the next.

CraFT proposes an implied loop notation enabling to specify several time steps in one line.

Implied loops are specified at the beginning of a line (before time value specification). Two notations are possible:

- an integer value enclosed by `:` characters specifying the number of implied loops between the time step of the preceding line (not inclueded) and the time step of the current line (included),
- a float value enclosed by `%` characters specifying the implied time steps between the time of the preceding line and the time step of the current line.

The time values t and the "loading modulus" k of the so-created loading steps are supposed to be linearly interpolated between their value in previous line ant in the current line; the directions d of the so-created loading steps are supposed to equal to the one in the current line.

For example:

```
#-----------------------------------
# prescribed strain
D
#-----------------------------------
# loading
#      t          direction       k
#             11 22 33 12 13 23
#- - - - - - - - - - - - - - - - - -
:10:   1.    1. 0. 0. 0. 0. 0.   10.
#
```

will create 10 loading steps form $t = 0.1$ to $t = 1.$, with a modulus k varying form $k = 1.$ to $k = 10.$ (as the previous time step is implicitely considered as $t = 0$ and $k = 0$).

It would have been equivalently written as:

```
#-----------------------------------
# prescribed strain
D
#-----------------------------------
# loading
```

```
#      t          direction         k
#             11 22 33 12 13 23
#- - - - - - - - - - - - - - - - - - - - -
       0.1   1. 0. 0. 0. 0. 0.    1.
       0.2   1. 0. 0. 0. 0. 0.    2.
       0.3   1. 0. 0. 0. 0. 0.    3.
       0.4   1. 0. 0. 0. 0. 0.    4.
       0.5   1. 0. 0. 0. 0. 0.    5.
       0.6   1. 0. 0. 0. 0. 0.    6.
       0.7   1. 0. 0. 0. 0. 0.    7.
       0.8   1. 0. 0. 0. 0. 0.    8.
       0.9   1. 0. 0. 0. 0. 0.    9.
       1.0   1. 0. 0. 0. 0. 0.   10.
```

or:

```
#-------------------------------------
# prescribed strain
D
#-------------------------------------
# loading
#      t           direction     k
#             11 22 33 12 13 23
#- - - - - - - - - - - - - - - - - - - - -
%0.1%  1.    1. 0. 0. 0. 0. 0.   10.
#
```

as implied-time step is equal to 0.1, the number of implied loops between $t = 0$ (preceding line) and $t = 1$ (current line) is $1/0.1 = 10$

Examples of loading file are given in C.1.

3.6 Output specifications

Output specification files contain lines beginning with a keyword, which can be composed of several words, followed by a = and one or several arguments.

Availables keywords are:

- `generic name`
- *xxx* `image` (where *xxx* is the name of a mechanical variable (stress, strain, ...) to be stored as an image
- *xxx* `moment` (where *xxx* is the name of a mechanical variable (stress, strain, ...) whose first and second moments have to be calculated and stored for each phase.

3.6.1 keyword: `generic name`

Argument following `generic name` is to be the lexical root of the names of all output files. For example, if output spec file contains line:

```
generic name=foo
```

craft will build a `foo.res` file to contain macroscopic results at each steps of the loading path, a `foo.perf` file to display statistics about execution, etc ...

3.6.2 keyword: *xxx* `image`

If the argument of keyword *xxx* `image` is `yes` (or if no argument is given), image(s) of *xxx* field are to be created. If argument is `no`, no images are created. *xxx* is the name of a mechanical variable; it can be common to all possible mechanical behaviors; i.e. `strain` and `stress`, or it can be specific to a given behavior.

A list of variables available for image storing is given for every behavior (see appendix B).

If the concerned mechanical variable is a second order tensor, an image of each 6 components of the tensor field are to be created; the name of these images is build with the "generic name" followed by `_t=` and the time (in the loading path) at which the image has been captured, followed by the name of the variable, and finally by the number of the component (`11`, `22`, `33`, `12`, `13`, or `23`).

For example, the following output spec file:

```
generic name=foo
stress image=yes
```

will create images of the 6 components of the stress field at the last time step of the loading path (let us say, at time 1s):

`foo_t=01.00000000e+00_stress11.ima`,

`foo_t=01.00000000e+00_stress22.ima`,

`foo_t=01.00000000e+00_stress33.ima`,

`foo_t=01.00000000e+00_stress12.ima`,

`foo_t=01.00000000e+00_stress13.ima`,

`foo_t=01.00000000e+00_stress23.ima`.

If just `yes` is given argument, the image(s) is stored at the last step of the loading path.

Moreover one can give the time(s) at which images are to be stored by entering a list of time specifications separated by commas.

Time values can be entered either by their actual value (in seconds) or by the number of their step in the loading path, in which case this number is entered as an integer value preceded by an at sign character (`@`).

The time value of the first step of the loading path can be specified by `firt` or by `begin` (or by its actual time value).

The time value of the last step of the loading path can be specified by `last` or `end` (or by its actual time value).

When two time specifications are separated by a colon character (`:`), images are to be stored at each step of the loading path between these two extreme time values. If a second colon character is entered and followed by a time value, this last value is taken as a time step.

Examples:

```
strain image = yes 10.,20, 30.:40.:@2, 45.:@100, @200
```

requires to store images of the strain tensor at times: $t = 10s$, $t = 20s$, once at every two time steps between $t = 30s$ and $t = 40s$, at every time steps between $t = 45s$ and the 100^{th} steps, and at the 200^{th} step of the loading path.

```
stress image=yes first:last
```

requires to store images of the stress tensor at every steps of the loading path (i.e. from the fisrt step to the last one).

```
stress image=yes 180.:last:@2
```

requires to store images of the stress tensor once at every two steps, from time $t = 180s$ to the last step of the loading path.

3.6.3 keyword: `im_format`

The format of the images which have to be stored as results of the computation can be specified by keyword `im_format`.

- `im_format=vtk`

  simple legacy VTK file format is prescribed

- `im_format=i3d`

  CraFT image format is prescibed

- `im_format=all`

  every image to be stored will be saved under both formats (VTK and i3d).

3.6.4 keyword: *xxx* `moment`

First and second moments of *xxx* variable has to be stored during calculation.

The syntax is similar to the one for image storage. The only difference, is that a sole file will be created for each variable whose moments are required to be stored, even if several times for storage are given.

Example:

```
generic name=foo
strain moment = yes 10.:20
```

will create a file named `foo_strain.mom` containing the first and second moments of the strain field in every phase, at every time steps between 10s and 20s.

3.7 Specification of reference material C_O

The reference material C_0 can be chosen:

- either automatically by entering: `auto` keyword (recommended)
- or explicitely by entering: `param` keyword followed the two Lamé coefficient of C_0 (C_0 is an istropic linear elastic material).

3.8 Required accuracy

The numerical method implemented in CraFT use an iterative process at each step of the loading path. Convergence is assumed to be reached when:

1 the modulus of the divergence of the stress field is lower than a given value,

2 the loading conditions are verified.

The modulus of the divergence of the stress field is computed in Fourier space as:

$$\|div(\boldsymbol{\sigma})\| = \sqrt{\sum_{\boldsymbol{\xi}} |\boldsymbol{\xi}.\hat{\boldsymbol{\sigma}}(\boldsymbol{\xi})|^2}$$

and is compared to a value entered by the user. Convergence is assumed to be reached when:

$$\|div(\boldsymbol{\sigma})\| \ < \ required\_accuracy\_for\_divergence\_of\_stress$$

Basically, the iterative process enables to prescribe macroscopic strain by forcing the strain field in Fourier space at null frequency to a given value.

Nevertheless, it is possible to prescribe macroscopic stress or to prescribe the direction of macroscopic stress via a secondary iterative scheme which proposes,

at each iteration, a new macroscopic strain which is then imposed to the null frequency of the strain field. Thus, it has to be verified, at each iteration, if prescribed loading conditions has been reached or not.

In the case of prescribed macroscopic stress, the iterative scheme is the following:

$$\mathbf{E}^{i+1} = \mathbf{E}^{i} + {\mathbf{C_0}}^{-1} : (\mathbf{\Sigma} - < \boldsymbol{\sigma}^{i} >)$$

where:

- $\mathbf{E}^{i}$ is the macroscopic strain at iteration i
- $\mathbf{\Sigma}$ is the prescribed macroscopic stress
- $< \boldsymbol{\sigma}^{i} >$ is the overall mean of the stress field at iteration i
- ${\mathbf{C_0}}^{-1}$ is the stiffness of reference material $\mathbf{C_0}$

and the convergence condition is:

$$\frac{\|\mathbf{\Sigma} - < \boldsymbol{\sigma}^{i} > \|}{\|\mathbf{\Sigma}\|} \quad < \quad required\_accuracy\_for\_loading\_conditions$$

In the case of prescribed direction of macroscopic stress, the iterative scheme is the following:

$$\begin{aligned} {\mathbf{C_0}}^{-1} : \mathbf{E}^{i+1} - k^{i+1}\mathbf{\Sigma_0} &= {\mathbf{C_0}}^{-1} : \mathbf{E}^{i} - < \boldsymbol{\sigma}^{i} > \\ \mathbf{E}^{i+1} : \mathbf{\Sigma_0} &= E(t) \end{aligned}$$

where:

- $\mathbf{\Sigma_0}$ is the prescribed direction of macroscopic stress
- $E(t)$ is the prescribed macroscopic strain in $\mathbf{\Sigma_0}$ direction (it is a scalar),
- $< \boldsymbol{\sigma}^{i} >$ is the overall mean of the stress field at iteration i
- k^{i+1} is a scalar to be computed

and the convergence condition is:

$$\frac{\|\mathbf{k^i} : \mathbf{\Sigma_0} - < \boldsymbol{\sigma}^{i} > \|}{\|k^{i} : \mathbf{\Sigma_0}\|} \quad < \quad required\_accuracy\_for\_loading\_conditions$$

So CraFT user has to enter two accuracy values for convergence of equilibrium (divergence of stress) and for convergence of loading condition; these two values has to be entered separated by a comma.

If the user enter just one value, it is applied to the two required accuracies.

4 Lexicon

In this section, we give the definition of some words or concepts often used all along this document, in the precise meaning that we have given to them.

4.1 Digital images

4.1.1 Generalities

A digital image is a set of physical points, called "pixels" in 2d and "voxels" in 3d (although the author of this document does not like this word and prefer to use "pixel" in 2d and in 3d), placed at the nodes of a regular grid of the space.

Thus, pixels are organized as a set of $n_1 \times n_2 \times n_3$ points, each pixel being separated from its previous neighbour along the k-th direction $(k = 1, 2, 3)$ by a given $\mathbf{p}_k$ vector.

Remark: a 2d image can be considered as a 3d image whose third direction has a 1 pixel depth ($n_3 = 1$).

Hence, the volume described in that way is a parallelepiped (and not necessarily a cube nor even a rectangular parallelepiped as it is usually defined) as $\mathbf{p}_k$ $(k = 1, 2, 3)$ vectors are not necessarily orthogonal nor having same magnitude.

With the definition of the position $\mathbf{s} = (s_1, s_2, s_3)$ of the first pixel in the list, the coordinate in the euclidian space $\mathbf{x} = (x_1, x_2, x_3)$ of each pixel can be got from its position in the digital image $\mathbf{i} = (i_1, i_2, i_3)$ (with $i_1 = 0, 1, ..., n_1 - 1$, $i_2 = 0, 1, ..., n_2 - 1$, $i_3 = 0, 1, ..., n_3 - 1$

$$\begin{aligned} \mathbf{x} &= \mathbf{s} + \textstyle\sum_{k=1,2,3} i_k \times \mathbf{p}_k \\ x_l &= s_l + \textstyle\sum_{k=1,2,3} i_k \times p_{kl} \quad (k = 1, 2, 3) \end{aligned}$$

(p_{kl} being the l-th component of $\mathbf{p}_k$ vector).

The data stored at each pixel could theoritically be of any kind: a scalar value, an integer value, a vector, a tensor, ...

In practice, it depends on the way images are implemented.

4.1.2 CraFT format of images

CraFT code proposes a C-structure of images called `CraftImage` whose pixels can contain:

- an integer value: `int`
- a scalar floating point: `float`
- a scalar floating point in double precision: `double`
- a vector (as a 3 dimension array in double precision): `double [3]`
- a symmetrical 2d order tensor (as a 6 dimension array in double precision): `double [6]`
- an array of double precision values of any dimension

CraFT proposes a format for image file which is (unfortunately) slightly different: values stored in pixels can only be scalars of type:

- signed 1-byte integer (`char`),
- unsigned 1-byte integer (`unsigned char`),
- signed integer (`int`),
- unsigned integer (`unsigned int`),
- floating point in simple precision (`float`)
- floating point in double precision (`double`)

That is why a `CraftImage` 2d order tensor image is stored into 6 different files, each one containing a given component of the tensor.

TO DO: homogenize image representations in CraFT between inside code and file format.

An CraFT image file is **binary** file consisting in a header (the size of which depends on the case= and in the set of pixel values (binary in IEEE 754 arithmetic).

The header comprised:

- 10 bytes describing the type of pixels the image contains:

  `HM2RS` : floating values in single precision (coded in 4 bytes)

  `HM2RD` : floating values in double precision (coded in 8 bytes)

  `HM2RI` : integer values

  `HM2RUI` : unsigned integer values

  `HM2RC` : character values

  `HM2RUC` : unsigned character values

`HMRS` : old (obsolete?) format for floating values in simple precision

- 20 bytes giving endianness of data values:

  `Big Endian` : data values are coded in big endian format

  `Little Endian` : data values are coded in little endian format

  Following data in the header are supposed to be coded following the endianness which has been declared here.

- header size (in bytes) : **only in old** `HMRS` **format** total size of the header
- n_1 n_2 n_3 : the number of pixels in the 3 directions, given as integer values coded in binary
- s_1 s_2 s_3 : the coordiantes of the first pixels of the image, given as double precision real values coded in binary
- p_{11} p_{12} p_{13} p_{21} p_{22} p_{23} p_{31} p_{32} p_{33} : the 3 components of the step vectores along the 3 directions, given as double precision real values coded in binary

  (Caution: In the case of old `HMRS` format, stpe vectors are supposed to be orthogonal, and just p_{11} p_{22} p_{33} are to be written here). the 3 components of the step vectores along the 3 directions, given as double precision real values coded in binary

Thus, except in the case of `HMRS` old format, the header comprises 138 bytes.

The pixel values are stored one after each other from the "first pixel" (i.e. the pixel with coordinates $\mathbf{x} = (s_1, s_2, s_3)$, $\mathbf{i} = (0, 0, 0)$) to the last, i_1 coordinate varying the fastest, and i_3 the slowest. In other words, pixels are stored in the following order: $\mathbf{i} = (0, 0, 0)$, $\mathbf{i} = (1, 0, 0)$, $\mathbf{i} = (2, 0, 0)$, .. $\mathbf{i} = (n_1 - 1, 0, 0)$, $\mathbf{i} = (0, 1, 0)$, $\mathbf{i} = (1, 1, 0)$, $\mathbf{i} = (2, 1, 0)$, .. $\mathbf{i} = (n_1 - 1, 1, 0)$, $\mathbf{i} = (0, 2, 0)$, $\mathbf{i} = (1, 2, 0)$, ... $\mathbf{i} = (n_1 - 1, 2, 0)$, ... $\mathbf{i} = (n_1 - 1, n_2 - 1, 0)$, $\mathbf{i} = (0, 0, 1)$, $\mathbf{i} = (1, 0, 1)$, ... $\mathbf{i} = (n_1 - 1, n_2 - 1, n_3 - 1)$

What some people could call Fortran-like indexing...

4.1.3 Simple legacy VTK file format

CraFT is able to read and write images formatted in simple legacy VTK format with:

- "STRUCTURED_POINTS" dataset
- pixels containing scalar values coded as float number.

Full details on simple legacy VTK format are given in:

`www.vtk.org/VTK/img/file-formats.pdf`

(this document being taken from the VTK User's Guide (published by Kitware Inc)).

In few words:

- VTK files consist in a man-readable header and a set of pixels,
- the pixels of a VTK file image are placed along the 3 axis of the cartesian grid (in other words: $\mathbf{p}_1 = (s_x, 0, 0)$, $\mathbf{p}_2 = (0, s_y, 0)$, $\mathbf{p}_3 = (0, 0, s_z)$ where s_x, s_y, s_z are the VTK spacings in the 3 directions,
- pixels can be stored either in ASCII or in binary format,
- pixels are stored in the same order as in CraFT format: pixels are ordered with x_1 increasing fastest, then x_2, then x_3,
- CraFT supposes that data in VTK files are represented in IEEE 754 floating point standard with big-endian byte ordering (to the opinion of the author of CraFT User's Guide, VTK format is not perfectly clear on how binary data has to be represented).

The main advantage of using VTK file format instead of Craft format is its much more common use. For example, images of this format can be visualized via well known 3D visualization programs such as Paraview and Mayavi2.

An example of a simple Legacy VTK image in ascii is given in 4.1

4.1.4 Conversion between CraFT format and simple legacy VTK file format

Two programs are available with craft distribution to convert from one format to the other:

- `i3dtovtk` : to convert from so-called"i3d" CraFT format to VTK format (either in binary or in ASCII format)
- `vtktoi3d` : to convert from VTK format to so-called"i3d" CraFT format

Type `i3dtovtk -h` and `vtktoi3d -h` in a unix terminal to get more details on how to use these programs.

```
# vtk DataFile Version 3.0
craft output
ASCII
DATASET STRUCTURED_POINTS
DIMENSIONS 32 32 1
ORIGIN 0.000000 0.000000 0.000000
SPACING  0.031250 0.031250 1.000000
POINT_DATA 1024
SCALARS scalars float
LOOKUP_TABLE default
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
0 0 0 0 0 2 2 2 2 2 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 0 2 2 2 2 2 2 2 2 2 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 0 2 2 2 2 2 2 2 2 2 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 2 2 2 2 2 2 2 2 2 2 2 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
0 0 2 2 2 2 2 2 2 2 2 2 2 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
0 0 2 2 2 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 2 2 2 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 2 2 2 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 2 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 2 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Figure 4.1: An example of a simple Legacy VTK image in ascii. This image contains two disks which are almost visible in the text file (although the image being seen upside down).

Appendix A How to run CraFT

Following examples shows different ways to run craft for the following problem:

- microsctructure described by image file: `micro01.ima`
- phases in microstructure described by file:`micro01.phases`
- material(s) in microstructure described by file: `micro01.mat`
- loading conditions described by file: `traction.dat`
- required ouptuts described by file: `micro01.output`
- reference material C_0 chosen by CraFT
- required precision: 1.10^{-4}

A.1 Case 1: interactive call

```
craft -i
Enter the name of the image of the characteristic function: micro01.ima
Enter the name of the file describing the phases: micro01.phases
Enter the name of the file describing the materials: micro01.mat
Enter the name of the file describing the loading conditions : traction.dat
Enter the name of the file describing required outputs: micro01.output
Enter C0:
          auto
          param   lb0 mu0
          matrix  21 coef Cij
auto
Enter required precision: 1.E-4
```

A.2 Case 2: inputs are described by configuration file `micro01a.in` in "without keywords" format

CraFT can be run using `micro01a.in` configuration file:

```
#-----------------------------------------------
# HM 30/12/2010
# file: micro01a.in
#
#-----------------------------------------------
# image file describing the microstructure:
micro01.ima
```

```
# file describing the different phases in the
# microstructure:
micro01.phases

# file describing the mechanical behavior of
# the different components of the material:
micro01.mat

# file describing loading conditions:
traction.dat

# file describing the selected outputs:
micro01.output

# choice of C0:
auto

# required precision:
1.E-4
#-------------------------------------------------
```

by typing following command line:

```
craft -f micro01a.in
```

or, alternately, by typing:

```
craft < micro01a.in
```

A.3 Case 3: inputs are described by configuration file `micro01b.in` in "keywords format"

CraFT can be run using `micro01b.in` configuration file:

```
#-------------------------------------------------
# HM 30/12/2010
# file: micro01b.in
#
#-------------------------------------------------
# file describing the mechanical behavior of the
# different components of the material:
Materials=micro01.mat

# file describing the different phases in the
# microstructure:
Phases=micro01.phases
```

```
# image file describing the microstructure:
Microstructure=micro01.ima

# file describing loading conditions:
loading=traction.dat

# choice of C0:
C0=auto

# required precision:
precision = 1.E-4

# file describing the selected outputs:
output = micro01.output

#-------------------------------------------------
```

by typing following command line:

```
craft -f micro01b.in
```

A.4 Case 4: inputs are described by configuration file `micro01c.in` in "keywords format", loading and output specified directly in the input file (instead of being described by files

```
#-------------------------------------------------
# HM 30/12/2010
# input file micro01c.in
#-------------------------------------------------
# file describing the mechanical behavior of the
# different components of the material:
Materials=micro01.mat

# file describing the different phases:
Phases=micro01.phases

# image file describing the microstructure:
Microstructure=micro01.ima

# loading conditions:
loading {
S
1.  1 0 0 0 0 0 1.
}
```

```
# choice of C0:
C0=auto

# required precision:
precision = 1.E-4

# file describing the selected outputs:
output {
generic name=micro01c
stress image = yes
strain image = no
}
#-------------------------------------------------
```

A.5 Case 5: problem specifications described one by one in a command line

```
craft -c micro01.ima -p micro01.phases -m micro01.mat \
-l traction.dat -o micro01.output -C auto -e 1.E-4
```

Appendix B File describing materials in CraFT

The first line of every given material specification consists in the identifier of this material (it is a integer value defining uniquely a given material) followed by the number describing its behavior (see table 3.2).

B.1 How to describe a void material

Behavior identifier: 0

No parameters to be entered for void materials.

Example:

```
#-------------------------------------------------------
# material nr 17 is a void material
17   0
#no further parameters are required
#-------------------------------------------------------
```

B.2 How to describe an isotropic linear elastic material

Behavior identifier: 1

Isotropic linear elasticity in CraFT is described by two parameters entered, one per line, in that order:

- Young's modulus
- Poisson coefficient

Example:

```
#-------------------------------------------------------
# material nr 32 is an isotropic linear elastic material
32   1
# Young's modulus:
10.
# Poisson coefficient:
0.23
#-------------------------------------------------------
```

B.3 How to describe an anisotropic linear elastic material

Behavior identifier: 3

A linear elastic (anisotropic or isotropic) material in CraFT is specified by an integer value in the range 0 to 4 to choose how the material specification will be entered, followed by the material specification.

The different possible cases are:

- case 0: the full stiffness matrix has to be entered
- case 1: isotropic case
- case 2: cubic symmetry
- case 3: hexagonal symmetry
- case 4: orthotropic symmetry

B.3.1 case 0: stiffness matrix entirely specified

The stiffness matrix has to be entered by its upper triangular part using Kelvin notations. I.e. if the stress tensor $\boldsymbol{\sigma}$ is represented as a vector s of 6 components: `s(i) with i=1,2,3,4,5 or 6` with:

$$\begin{aligned}
\sigma_1 &= \sigma_{11} \\
\sigma_2 &= \sigma_{22} \\
\sigma_3 &= \sigma_{33} \\
\sigma_4 &= \sqrt{2}\sigma_{23} \\
\sigma_5 &= \sqrt{2}\sigma_{13} \\
\sigma_6 &= \sqrt{2}\sigma_{12}
\end{aligned}$$

and if the strain tensor $\boldsymbol{\varepsilon}$ s represented by a 6 component vector e:

$$\begin{aligned}
\varepsilon_1 &= \varepsilon_{11} \\
\varepsilon_2 &= \varepsilon_{22} \\
\varepsilon_3 &= \varepsilon_{33} \\
\varepsilon_4 &= \sqrt{2}\varepsilon_{23} \\
\varepsilon_5 &= \sqrt{2}\varepsilon_{13} \\
\varepsilon_6 &= \sqrt{2}\varepsilon_{12}
\end{aligned}$$

The stiffness tensor can be represented as the matrix C as follows:

$$\begin{pmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \sigma_4 \\ \sigma_5 \\ \sigma_6 \end{pmatrix} = \begin{pmatrix} C_{11} & C_{12} & C_{13} & C_{14} & C_{15} & C_{16} \\ C_{21} & C_{22} & C_{23} & C_{24} & C_{25} & C_{26} \\ C_{31} & C_{32} & C_{33} & C_{34} & C_{35} & C_{36} \\ C_{41} & C_{42} & C_{43} & C_{44} & C_{45} & C_{46} \\ C_{51} & C_{52} & C_{53} & C_{54} & C_{55} & C_{56} \\ C_{61} & C_{62} & C_{63} & C_{64} & C_{65} & C_{66} \end{pmatrix} \cdot \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \varepsilon_4 \\ \varepsilon_5 \\ \varepsilon_6 \end{pmatrix}$$

or:

$$\begin{pmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{33} \\ \sqrt{2}\sigma_{23} \\ \sqrt{2}\sigma_{13} \\ \sqrt{2}\sigma_{12} \end{pmatrix} = \begin{pmatrix} C_{11} & C_{12} & C_{13} & C_{14} & C_{15} & C_{16} \\ C_{21} & C_{22} & C_{23} & C_{24} & C_{25} & C_{26} \\ C_{31} & C_{32} & C_{33} & C_{34} & C_{35} & C_{36} \\ C_{41} & C_{42} & C_{43} & C_{44} & C_{45} & C_{46} \\ C_{51} & C_{52} & C_{53} & C_{54} & C_{55} & C_{56} \\ C_{61} & C_{62} & C_{63} & C_{64} & C_{65} & C_{66} \end{pmatrix} \cdot \begin{pmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ \varepsilon_{33} \\ \sqrt{2}\varepsilon_{23} \\ \sqrt{2}\varepsilon_{13} \\ \sqrt{2}\varepsilon_{12} \end{pmatrix}$$

The upper triangular part of the matrix is entered into CraFT like that:

$$\begin{matrix} C_{11} & C_{12} & C_{13} & C_{14} & C_{15} & C_{16} \\ & C_{22} & C_{23} & C_{24} & C_{25} & C_{26} \\ & & C_{33} & C_{34} & C_{35} & C_{36} \\ & & & C_{44} & C_{45} & C_{46} \\ & & & & C_{55} & C_{56} \\ & & & & & C_{66} \end{matrix}$$

Examples:

```
#-------------------------------------------------------
# behavior of material number 15 is anisotropic linear
# elastic (3):
15 3
# its stiffness matrix will be entered:
0
# stiffness matrix:
13930.   7082   5765     0.     0.     0.
       13930.  5765     0.     0.     0.
              15010.    0.     0.     0.
                     6028.     0.     0.
                            6028.     0.
                                   6828.
#-------------------------------------------------------
```

B.3.2 case 1: isotropic case

In this case, the behavior of material is supposed to be isotropic linear elasticity. The user has to enter:

- the Young's modulus: E
- the Poisson coefficient: ν

The stiffness matrix is the calculated as:

$$\begin{pmatrix} \lambda+2\mu & \lambda & \lambda & 0 & 0 & 0 \\ \lambda & \lambda+2\mu & \lambda & 0 & 0 & 0 \\ \lambda & \lambda & \lambda+2\mu & 0 & 0 & 0 \\ 0 & 0 & 0 & 2\mu & 0 & 0 \\ 0 & 0 & 0 & 0 & 2\mu & 0 \\ 0 & 0 & 0 & 0 & 0 & 2\mu \end{pmatrix}$$

with λ and μ being the Lamé coefficient:

$$\begin{aligned} \lambda &= \frac{E.\nu}{(1+\nu).(1-2\nu)} \\ \mu &= \frac{E}{2.(1+\nu)} \end{aligned}$$

Example:

```
#-----------------------------------------------------------
# material nr 32 is an linear elastic material
32   3
# ... and it is isotropic:
1
# Young's modulus:
10.
# Poisson coefficient:
0.23
#-----------------------------------------------------------
```

TO DO: This is redundant with behavior number 1, except that the full stiffness matrix is calculate and then used to apply the behavior law instead of using Lamé coefficients.

B.3.3 case 2: cubic symmetry

In the case cubic symmetry, the user has to enter the following parameters:

- bulk modulus K
- μ_1
- μ_2

the stffness matrix being then calculated as:

$$\begin{pmatrix} (3K+4\mu_1)/3 & (3K-2\mu_1)/3 & (3K-2\mu_1)/3 & 0 & 0 & 0 \\ (3K-2\mu_1)/3 & (3K+4\mu_1)/3 & (3K-2\mu_1)/3 & 0 & 0 & 0 \\ (3K-2\mu_1)/3 & (3K-2\mu_1)/3 & (3K+4\mu_1)/3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2\mu_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2\mu_2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2\mu_2 \end{pmatrix}$$

B.3.4 case 3: hexagonal symmetry

In the case hexagonal symmetry, the user has to enter the following parameters:

- bulk modulus K
- μ_t
- μ_l
- E_l
- ν_l

the stffness matrix being then calculated as:

$$\begin{pmatrix} K+\mu_t & K-\mu_t & 2\nu_l K & 0 & 0 & 0 \\ K-\mu_t & K+\mu_t & 2\nu_l K & 0 & 0 & 0 \\ 2\nu_l K & 2\nu_l K & E_l + 4\nu_l^2 K & 0 & 0 & 0 \\ 0 & 0 & 0 & 2\mu_l & 0 & 0 \\ 0 & 0 & 0 & 0 & 2\mu_l & 0 \\ 0 & 0 & 0 & 0 & 0 & 2\mu_t \end{pmatrix}$$

B.3.5 case 4: orthotropic symmetry

In the case of orthotropic symmetry, the user has to enter the following parameters:

- 3 Young'moduli: E_1, E_2, E_3
- 3 Poisson coefficients: ν_{12}, ν_{13}, ν_{23}
- 3 shear moduli: μ_{12}, μ_{13}, μ_{23}

the stiffness matrix being then calculated as

$$\begin{pmatrix} D.\frac{1-\nu_{23}^2 E_3/E_2}{E_2 E_3} & D.\frac{\nu_{12}E_2/E_1+\nu_{13}\nu_{23}E_3/E_1}{E_2E_3} & D.\frac{\nu_{13}E_3/E_1+\nu_{12}\nu_{23}E_3/E_1}{E_1E_2} & 0 & 0 & 0 \\ & D.\frac{1-\nu_{13}.\nu_{13}.E_3/E_1}{E_1E_3} & D.\frac{\nu_{23}.E_3/E_2+\nu_{12}.\nu_{13}.E_3/E_1}{E_1E_3} & 0 & 0 & 0 \\ & & D.\frac{1-\nu_{12}\nu_{12}E_2/E_1}{E_1E_2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 2\mu_{23} & 0 & 0 \\ 0 & 0 & 0 & 0 & 2\mu_{13} & 0 \\ 0 & 0 & 0 & 0 & 0 & 2\mu_{12} \end{pmatrix}$$

with:

$$D = E_1.E_2.E_3/(1-\nu_{23}\nu_{23}E_3/E_2 - \nu_{13}\nu_{13}E_3/E_1 - \nu_{12}\nu_{12}E_2/E_1 - 2\nu_{12}\nu_{13}\nu_{23}E_3/E_1)$$

or:

$$\begin{pmatrix} E_1(1-\nu_{23}\nu_{32})/k & E_1(\nu_{23}\nu_{31}+\nu_{21})/k & E_1(\nu_{21}\nu_{32}+\nu_{31})/k & 0 & 0 & 0 \\ E_2(\nu_{13}\nu_{32}+\nu_{12})/k & E_2(1-\nu_{13}\nu_{31})/k & E_2(\nu_{12}\nu_{31}+\nu_{32})/k & 0 & 0 & 0 \\ E_3(\nu_{12}\nu_{23}+\nu_{13})/k & E_3(\nu_{13}\nu_{21}+\nu_{23})/k & E_3(1-\nu_{12}\nu_{21})/k & 0 & 0 & 0 \\ 0 & 0 & 0 & 2\mu_{_{23}} & 0 & 0 \\ 0 & 0 & 0 & 0 & 2\mu_{13} & 0 \\ 0 & 0 & 0 & 0 & 0 & 2\mu_{12} \end{pmatrix}$$

with:

$$k = 1 - \nu_{23}\nu_{32} - \nu_{12}\nu_{21} - \nu_{13}\nu_{31} - \nu_{12}\nu_{23}\nu_{31} - \nu_{21}\nu_{32}\nu_{13}$$

and:

$$\nu_{21} = E_2/E_1.\nu_{12} \ , \ \nu_{32} = E_3/E_2.\nu_{23} \ , \ \nu_{31} = E_3/E_1.\nu_{13}$$

B.4 How to describe an elastic-perfectly-plastic Von Mises material

Behavior identifier: 2

In the case of an elastic-perfectly-plastic material with von Mises yield criterion (the linear elastic part being supposed to isotropic) the user has to enter the following parameters:

- Young's modulus
- Poisson coefficient
- Yield stress

The algorithm used is the radial return algorithm.

Example:

```
#-----------------------------------------------------------
# material nr 8 is an elastic-perfectly-plastic material
8   2
# Young's modulus:
10.
# Poisson coefficient:
0.23
# Yield stress:
1.2
#-----------------------------------------------------------
```

B.5 How to describe an elastic-plastic Von Mises material

Behavior identifier: 4

In the case of an elastic-plastic material with von Mises yield criterion and hardening (the linear elastic part being supposed to isotropic) the user has to enter the following parameters:

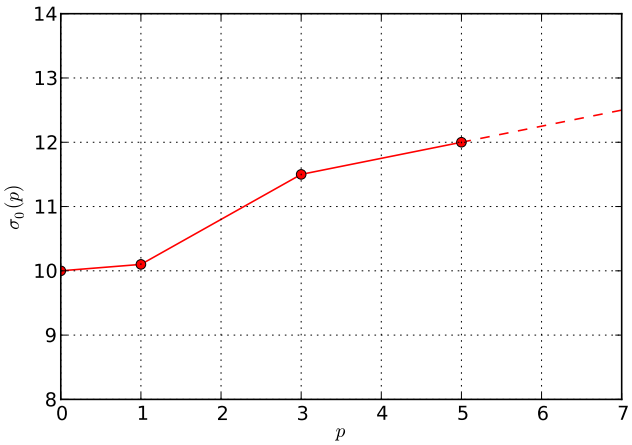
- Young's modulus
- Poisson coefficient
- A flag telling the hardening type:

  0: without hardening $\sigma_0(p) = y_s$ (redundant with Sec. B.4). The user has then to enter the yield stress y_s.

  1: with linear hardening $\sigma_0(p) = y_s + Hp$. The yield stress ys and then the plastic modulus H have to be provided.

  2: with tabulated values of the hardening. The dependance of the isotropic hardening σ_0 with respect to the cumulated plastic strain p is specified through a table. The user has to give the number of points of the table, then a list of pairs of points defining the isotropic hardening criterion, for example:

```
# Number of points
4
# Pairs of points
# (p, sigma0)
0. 10.0
1. 10.1
3. 11.5
5. 12.0
```



  with extrapolation beyond the last pair based on the last segment.

The algorithm used is the radial return algorithm.

Example:

```
#-----------------------------------------------------------
# material nr 8 is an elastic-plastic material
# without linear hardening
8   4
# Young's modulus:
10.
# Poisson coefficient:
0.23
# Yield stress:
10
# Plastic modulus:
0.3
#-----------------------------------------------------------
```

Appendix C Examples

C.1 Examples of loading files

C.1.1 example of creep loading

In the following example of loading specification file, creep loading is prescribed: macroscopic stress is prescribed (C in the first directive) to be $\Sigma_{11} = 10$, $\Sigma_{ij\neq 11} = 0$ at time $t = 0.1s$ and following time steps.

```
#-----------------------------------
# prescribed stress
C
#-----------------------------------
# loading
#      t          direction        k
#             11 22 33 12 13 23
#- - - - - - - - - - - - - - - - - - -
       0.1   1. 0. 0. 0. 0. 0.    10.
       0.2   1. 0. 0. 0. 0. 0.    10.
       0.3   1. 0. 0. 0. 0. 0.    10.
       0.4   1. 0. 0. 0. 0. 0.    10.
       0.5   1. 0. 0. 0. 0. 0.    10.
       0.6   1. 0. 0. 0. 0. 0.    10.
       0.7   1. 0. 0. 0. 0. 0.    10.
       0.8   1. 0. 0. 0. 0. 0.    10.
       0.9   1. 0. 0. 0. 0. 0.    10.
       1.0   1. 0. 0. 0. 0. 0.    10.
```

A more concise specification using implied loop notation would be:

```
#-----------------------------------
# prescribed stress
C
#-----------------------------------
# loading
#      t          direction        k
#             11 22 33 12 13 23
#- - - - - - - - - - - - - - - - - - -
       0.1   1. 0. 0. 0. 0. 0.    10.
:9:    1.0   1. 0. 0. 0. 0. 0.    10.
```

Remarks:

- The line for time $t = 0.1$ is necessary as CraFT implies that macroscopic stress is null at time $t = 0$.

- The number of implied loops between $t = 0.1$ (not included) and $t = 1$ (included) is 9, thus the implied time step is: $0.1 = (1 - 0.1)/9$.

Another possibility for the same loading conditions would be:

```
#-----------------------------------
# prescribed stress
C
#-----------------------------------
# loading
#      t          direction        k
#            11 22 33 12 13 23
#- - - - - - - - - - - - - - - - - - -
       0.1   1. 0. 0. 0. 0. 0.    10.
%0.1%  1.0   1. 0. 0. 0. 0. 0.    10.
```

Time steps of implied loops between $t = 0.1$ and $t = 1$ being prescribed to 0.1

C.1.2 example of simple traction

In this example, a simple traction is applied in 11 direction ($\Sigma_{11} \neq 0\ \Sigma ij \neq 11 = 0$) untill $\mathbf{E} : \mathbf{d} = 50$ (in other words, untill $E_{11} = 50$) at time $t = 20$; 1000 time steps are applied.

```
#-----------------------------------
# prescribed direction of stress
S
#-----------------------------------
# loading
#         t          direction        k
#               11 22 33 12 13 23
#- - - - - - - - - - - - - - - - - - -
:1000:    20.   1. 0. 0. 0. 0. 0.    50.
```