# HyperDrive documentation

G.T. Houlsby, September 2020

## Contents

# 1 Introduction

HyperDrive is a Python script that implements hyperplasticity based models. The script has been developed in the Spyder environment, but will probably run correctly in other Python environments.

In the following, all text in *italics* should be substituted by strings or numbers as appropriate. Optional arguments are enclosed in square brackets [ ].

In its most basic form Hyperdrive requires just two additional supporting files:

- A further Python script "*model_file*.py" that provides definitions for the basic functions of hyperplasticity theory for a particular constitutive model (see Section 3).
- A data file "*input_file*.dat" that specifies the test to be simulated (see Section 4).

Hyperdrive offers the following options for analyses:

- Rate independent or rate-dependent analysis
- Analysis based on the *f* and *y* functions (*f* and *w* for rate dependent) or the *g* and *y* functions (*g* and *w* for rate dependent).
- Derivatives based either on analytical expressions provided by the user, or determined automatically using the "autograd" package, or determined numerically.
- Strain control, stress control or (for multidimensional models) mixed control.

HyperDrive uses a number of standard Python packages, which will need to be installed on your system. These include "copy", "importlib", "matplotlib", "os", "re", "scipy", "sys" and "time", although a number of these are not central to its use, and if not available there would be simple workarounds. However, it makes very heavy use of "numpy". If "autograd" is used the version of "numpy" used should be "autograd.numpy", but if "autograd" is not available the simple "numpy" package can be used (and the line "auto = True" at the beginning of HyperDrive changed to "auto = False"). Within "numpy" the function most used is "einsum", which is central to the entire HyperDrive package.

Output is available in tabulated form (also as a .csv file) and as a variety of different plots (also as .png files).

# 2 Running HyperDrive

**Step 1**: In the Spyder environment, with HyperDrive.py in the appropriate directory, **either**:

Open HyperDrive.py in the editing window and then run the script, **or**:

In the console window type "`from HyperDrive import *`".

In other environments there will be similar procedures. At this stage you should see a message in the console window stating that the HyperDrive routines have been loaded.

**Step 2**: In the console window type "`drive()`", or "`drive(`"*input_file*"`)`".

On starting HyperDrive the user is prompted for name of "*input_file*". The ".dat" extension is assumed if not given. The file of course must be present in the appropriate directory. See also section 4.7 for how to access HyperDrive routines directly from Python without the need for an input file.

# 3 Specification of “*model_file*”

The Python script “*model_file*” contains a series of function definitions as follows:

| **Code** | **Notes** |
|---|---|
| `import autograd.numpy as np` | Necessary for almost all models |
| `from HyperDrive import Utils as hu` | Necessary if certain utilities are used |
| `file = “filename”` | Omit the .py extension |
| `name = “Brief description of model”` | |
| `mode = …` | |
| `ndim = …` | |
| `n_int = …` | |
| `n_y = …` | |
| [check_eps = …*code*…] | Optional value used by check() function |
| [check_sig = …*code*…] | Optional value used by check() function |
| [check_alp = …*code*…] | Optional value used by check() function |
| [check_chi = …*code*…] | Optional value used by check() function |
| […*code*…] | Other utility functions as required |
| `def deriv():`…*code*… | Derives certain necessary further values |
| `[def f(eps,alp):` …*code*…] | Implements $f = f(\varepsilon,\alpha)$ |
| `[def dfde(eps,alp):` …*code*…] | Implements $df/d\varepsilon$ |
| `[def dfda(eps,alp):` …*code*…] | Implements $df/d\alpha$ |
| `[def d2fdede(eps,alp):` …*code*…] | Implements $d^2f/d\varepsilon d\varepsilon$ |
| `[def d2fdeda(eps,alp):` …*code*…] | Implements $d^2f/d\varepsilon d\alpha$ |
| `[def d2fdade(eps,alp):` …*code*…] | Implements $d^2f/d\alpha d\varepsilon$ |
| `[def d2fdada(eps,alp):` …*code*…] | Implements $d^2f/d\alpha d\alpha$ |
| `[def g(sig,alp)` …*code*…] | Implements $g = g(\sigma,\alpha)$ |
| `[def dgds(sig,alp):` …*code*…] | Implements $dg/d\sigma$ |
| `[def dgda(sig,alp):` …*code*…] | Implements $dg/d\alpha$ |
| `[def d2gdsds(sig,alp):` …*code*]… | Implements $d^2g/d\sigma d\sigma$ |
| `[def d2gdsda(sig,alp):` …*code*…] | Implements $d^2g/d\sigma d\sigma$ |
| `[def d2gdads(sig,alp):` …*code*…] | Implements $d^2g/d\sigma d\sigma$ |
| `[def d2gdada(sig,alp):` …*code*…] | Implements $d^2g/d\sigma d\sigma$ |
| `[def d_f(alpdot,eps,alp):` …*code*…] | Implements $d^f = d^f(\dot{\alpha},\varepsilon,\alpha)$ |
| `[def y_f(chi,eps,alp):` …*code*…] | Implements $y^f = y^f(\chi,\varepsilon,\alpha)$ |
| `[def dydc_f(chi,eps,alp):` …*code*…] | Implements $dy^f/d\chi$ |
| `[def dyde_f(chi,eps,alp):` …*code*…] | Implements $dy^f/d\varepsilon$ |
| `[def dyda_f(chi,eps,alp):` …*code*…] | Implements $dy^f/d\alpha$ |
| `[def d_g(alpdot,eps,alp):` …*code*…] | Implements $d^g = d^g(\dot{\alpha},\sigma,\alpha)$ |
| `[def y_g(chi,sig,alp):` …*code*…] | Implements $y^g = y^g(\chi,\sigma,\alpha)$ |
| `[def dydc_g(chi,sig,alp):` …*code*…] | Implements $dy^g/d\chi$ |

| | |
|---|---|
| `[def dyds_g(chi,sig,alp):` …*code*…`]` | Implements $dy^g / d\sigma$ |
| `[def dyda_g(chi,sig,alp):` …*code*…`]` | Implements $dy^g / d\alpha$ |
| `[def w_f(chi,eps,alp):` …*code*…`]` | Implements $w^f = w^f(\chi,\varepsilon,\alpha)$ |
| `[def dwdc_f(chi,eps,alp):` …*code*…`]` | Implements $dw^f / d\chi$ |
| `[def w_g(chi,sig,alp):` …*code*…`]` | Implements $w^g = w^g(\chi,\sigma,\alpha)$ |
| `[def dwdc_f(chi,sig,alp):` …*code*…`]` | Implements $dw^f / d\chi$ |
| `[def update(t,eps,sig,alp,chi,`<br>`dt,deps,dsig,dalp,dchi):` …*code*…`]` | Update certain variables if necessary |
| `[def plot(rec, pname):` …*code*…`]` | Optional special code for plotting |
| deriv() | Ensure that derived values are obtained on loading |

At the start of the module code must set values of (at least) the following variables: `mode`, `const`, `n_y`, `n_int`.

Note that functions `f`, `g` are optional, as they are currently not used by HyperDrive unless automatic or numerical differentiation is used. They are used by the code checking routine check() (see section 7).

The functions `d_f` and `d_g` are optional, as they are currently not used by HyperDrive at all (but may be in the future).

Functions `w_f`, `dwdc_f`, `w_g` and `dwdc_g` are only required if the rate-dependent formulation is used. Conversely, `y_f`, `dydc_f`, `dyde_f`, `dyda_f`, `y_g`, `dydc_g`, `dyde_g` and `dyda_g` are only required if the rate-independent formulation is used.

Function "`plot`" is only required if special plots are used for this model.

Function "update" is only required for certain more complex models.

If only strain-controlled increments are used or the keyword "*f-form" is specified (see section 4), then only the functions `deriv`, `dfde`, `d2fdede`, `d2fdeda`, `d2fdade`, `d2fdada`, `y_f`, `dydc_f`, `dyde_f` and `dyda_f` are required.

Alternatively, if only stress-controlled increments are used or the keyword "*g-form" is specified (see Section 4), then only the functions `deriv`, `dgds`, `d2gdsds`, `d2gdsda`, `d2gdads`, `d2gdada`, `y_g`, `dydc_g`, `dyds_g` and `dyda_g` are required.

If any first or second derivative is not provided, automatic or numerical differentiation will be used to obtain the derivative (in which case the base function will of course be required). Preferences for which form of differentiation are specified by the "*prefs" keyword, with the default being: 1. Analytical (*i.e.* user supplied), 2. Automatic (using "autograd"), 3. Numerical. Analytical is normally the fastest (but requires more effort by the user), followed by numerical and then automatic. However, automatic is preferred by default as it may be more accurate than numerical.

## 3.1 Modes of operation

HyperDrive operates in three possible "modes".

- Mode 0: Strain and stress variables are scalars and are treated within Python each as a single variable (`eps` and `sig`). Allowance is made, however, for multiple internal variables and their corresponding generalised stresses (`alp` and `chi`), each of which is of the form `numpy.array(n_int)` where `n_int` is the number of internal variables. Allowance is made also for multiple yield surfaces, so that each yield function is of the form `numpy.array(n_y)` where `n_y` is the number of yield surfaces. For many simple models `n_y = n_int`, but allowance is made for the possibility that this is not the case.

- Mode 1: Strain and stress variables are vectors and are implemented as one dimensional arrays `numpy.array(ndim)`, where `ndim` is the dimensionality. Internal variables and their corresponding generalised stresses are of the form `numpy.array([n_int,ndim])`.
- Mode 2 (not yet fully implemented): Strain and stress variables are second order tensors and are implemented as two dimensional arrays `numpy.array([ndim,ndim])`. Internal variables and their corresponding generalised stresses are of the form `numpy.array([n_int,ndim,ndim])`.

The dimensionalities of the relevant variables for the different modes are given in Table 1.

**Table 1: dimensionality of principal variables**

| **Variables** | **Mode 0** | **Mode 1** | **Mode 2** |
|---|---|---|---|
| $f$ , $g$ , $d^f$ , $d^g$ , $w^f$ , $w^g$ | Scalar | Scalar | scalar |
| $\varepsilon$ , $\sigma$ , $\frac{\partial f}{\partial \varepsilon}$ , $\frac{\partial g}{\partial \sigma}$ | Scalar | array($n_{dim}$) | array($n_{dim}$, $n_{dim}$) |
| $\frac{\partial^2 f}{\partial \varepsilon \partial \varepsilon}$ , $\frac{\partial^2 g}{\partial \sigma \partial \sigma}$ | Scalar | array($n_{dim}$, $n_{dim}$) | array($n_{dim}$, $n_{dim}$, $n_{dim}$, $n_{dim}$) |
| $\alpha$ , $\chi$ , $\frac{\partial f}{\partial \alpha}$ , $\frac{\partial g}{\partial \alpha}$ , $\frac{\partial w^f}{\partial \alpha}$ , $\frac{\partial w^g}{\partial \alpha}$ | array($n_{int}$) | array($n_{int}$, $n_{dim}$) | array($n_{int}$, $n_{dim}$, $n_{dim}$) |
| $\frac{\partial^2 f}{\partial \varepsilon \partial \alpha}$ , $\frac{\partial^2 f}{\partial \alpha \partial \varepsilon}$ , $\frac{\partial^2 g}{\partial \sigma \partial \alpha}$ , $\frac{\partial^2 g}{\partial \alpha \partial \sigma}$ | array($n_{int}$) | array($n_{int}$, $n_{dim}$, $n_{dim}$) | array($n_{int}$, $n_{dim}$, $n_{dim}$, $n_{dim}$, $n_{dim}$) |
| $\frac{\partial^2 f}{\partial \alpha \partial \alpha}$ , $\frac{\partial^2 g}{\partial \alpha \partial \alpha}$ | array($n_{int}$, $n_{int}$) | array($n_{int}$, $n_{int}$, $n_{dim}$, $n_{dim}$) | array($n_{int}$, $n_{int}$, $n_{dim}$, $n_{dim}$, $n_{dim}$, $n_{dim}$) |
| $y^f$ , $y^g$ , $\Lambda^f$ , $\Lambda^g$ | array($n_y$) | array($n_y$) | array($n_y$) |
| $\frac{\partial y^f}{\partial \varepsilon}$ , $\frac{\partial y^g}{\partial \sigma}$ | array($n_y$) | array($n_y$, $n_{dim}$) | array($n_y$, $n_{dim}$, $n_{dim}$) |
| $\frac{\partial y^f}{\partial \chi}$ , $\frac{\partial y^g}{\partial \chi}$ , $\frac{\partial y^f}{\partial \alpha}$ , $\frac{\partial y^g}{\partial \alpha}$ | array($n_y$,$n_{int}$) | array($n_y$, $n_{int}$, $n_{dim}$) | array($n_y$, $n_{int}$, $n_{dim}$, $n_{dim}$) |

## 3.2 Example model file

An example model file for a simple Mode 0 model "h1epk" (see Section 6.1.3) that illustrates a number of features is given below. This implements a simple elastic-plastic model with linear kinematic hardening, for the case of a single stress and strain variable. Note particularly the dimensionality returned for each of the derivatives. For some simpler model files see Section XX.

```
import numpy as np
from HyperDrive import Utils as hu #necessary because use mac()

check_eps = 0.3
check_sig = 8.0
check_alp = np.array([0.2])
check_chi = np.array([-1.09])

file = "h1epk"
name = "1D Linear Elastic - Plastic with Kinematic Hardening"
mode = 0
```

```
ndim = 1
n_int = 1
n_y = 1
n_const = 3
name_const = ["E", "k", "H"]
const = [100.0, 1.0, 5.0]
mu = 0.1

def deriv():
    global E, k, H
    E = const[0]
    k = const[1]
    H = const[2]

deriv()

def f(eps,alp): return E*((eps-alp[0])**2)/2.0 + H*(alp[0]**2)/2.0
def dfde(eps,alp): return E*(eps-alp[0])
def dfda(eps,alp): return np.array([-E*(eps-alp[0]) + H*alp[0]])
def d2fdede(eps,alp): return E
def d2fdeda(eps,alp): return np.array([-E])
def d2fdade(eps,alp): return np.array([-E])
def d2fdada(eps,alp): return np.array([[E + H]])

def g(sig,alp): return -(sig**2)/(2.0*E) - sig*alp[0] + H*(alp[0]**2)/2.0
def dgds(sig,alp): return -sig/E - alp[0]
def dgda(sig,alp): return np.array([-sig + H*alp[0]])
def d2gdsds(sig,alp): return -1.0/E
def d2gdsda(sig,alp): return np.array([-1.0])
def d2gdads(sig,alp): return np.array([-1.0])
def d2gdada(sig,alp): return np.array([[H]])

def d_f(alpr,eps,alp): return k*abs(alpr)

def y_f(chi,eps,alp): return np.array([np.abs(chi[0]) - k])
def dydc_f(chi,eps,alp): return np.array([[hu.S(chi[0])]])
def dyde_f(chi,eps,alp): return np.array([0.0])
def dyda_f(chi,eps,alp): return np.array([[0.0]])

def d_g(alpr,eps,alp): return k*abs(alpr)

def y_g(chi,eps,alp): return np.array([np.abs(chi[0]) - k])
def dydc_g(chi,eps,alp): return np.array([[hu.S(chi[0])]])
def dyds_g(chi,sig,alp): return np.array([0.0])
def dyda_g(chi,sig,alp): return np.array([[0.0]])

def w_f(chi,eps,alp): return (hu.mac(abs(chi[0]) - k)**2)/(2.0*mu)
def dwdc_f(chi,eps,alp): return np.array([hu.S(chi[0])*hu.mac(abs(chi[0])-k)/mu])

def w_g(chi,eps,alp): return (hu.mac(abs(chi[0]) - k)**2)/(2.0*mu)
def dwdc_g(chi,eps,alp): return np.array([hu.S(chi[0])*hu.mac(abs(chi[0])-k)/mu])
```

# 4 Commands listed in “*input_file*”

Commands should be listed in a file “*input_file*.dat”.

# Any line beginning with # is treated as a comment and ignored.

Blank lines are also ignored.

## 4.1 Specifying the model

*title *title*

*mode *mode [ndim]*

Set mode:

*mode* = 0 – stress and strain variables are scalars (and *ndim* not required)
*mode* = 1 – stress and strain variables are vectors of length *ndim*
*mode* = 2 – stress and strain variables are tensors of dimension (*ndim, ndim*)

*model *model*

Functions and their derivatives will be as defined in file "*model*.py". The ".py" extension is assumed.

*const *constants...*

The constants used in the specified model. Different models require different numbers of constants see specifications of models in Section 6.

*tweak *const_name value*

Change the value of material constant with *const_name* to *value.*

*const_from_curve *modtype curve*

## 4.2 Options for analysis

*prefs *pref1 pref2 pref3*

Set the preferences for differentiation methods. Each of *pref1*, *pref2* and *pref3* should be one of "analytical", "automatic" or "numerical" (with no repetition). The default is:

*prefs analytical automatic numerical

Analytical differentiation mode (default first choice): all first and second differentials are evaluated analytically if possible from the user-supplied functions.

Automatic differentiation mode (default second choice): all first and second differentials are evaluated using "autograd". This produces relatively slow code, but means that the user does not have to supply the differentials of the base functions.

Numerical differentiation mode (default third choice): all first and second differentials are evaluated numerically.

If insufficient routines are supplied so that a differential cannot be determined by any of the above methods, then if that routine is needed the program will (obviously) fail. For example, the function "dfde" would either have to be supplied by the user, or the function "f" supplied which could be differentiated automatically or numerically.

*f_form

Use models derived from *f* (Helmholtz free energy)

*g_form

Use models derived from *g* (Gibbs free energy)

*rate [*mu*]

Use rate-dependent analysis. Optional *mu* value over-rides the default value in the model.

*rateind

Use rate-independent analysis (default)

*acc *acc*

Specify the acceleration factor used in yield surface correction (see Section 8 for rate-independent incremental algorithms).

## 4.3 Initialisation and flow control

*start

Start the test.

*restart

Restart a new test.

*init_stress *sig₁ [sig₂…ndim]*

*init_strain *eps₁ [eps₂…ndim]*

*end

Stop processing

## 4.4 Stress and strain increment commands

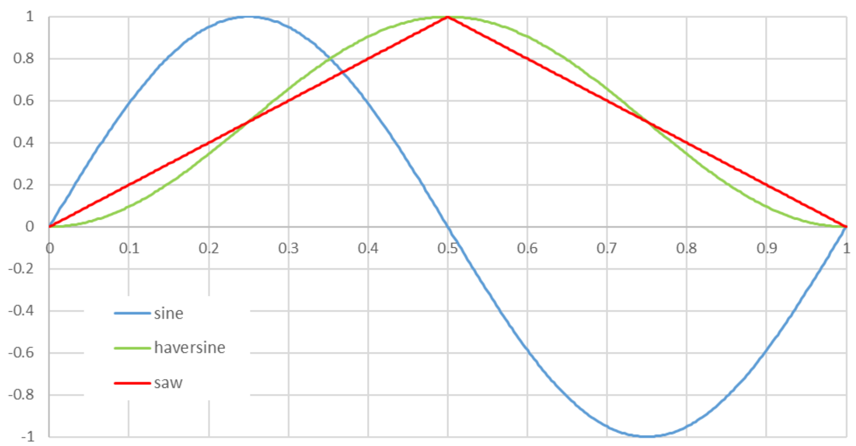
*general_inc *S E Tdt dt nprint nsub*

Use the control statement $S_{ij}d\sigma_j + E_{ij}d\varepsilon_j = T_i dt$ to define an increment. *S* and *E* give the terms in the matrices $S_{ij}$ and $E_{ij}$ and *Tdt* the vector $T_i dt$. The increment is divided into *nprint* printing (or plotting) points each calculated using *nsub* substeps. This option allows very versatile control of increments, including those in which (for instance) some directions are stress-controlled and other strain-controlled, or in which ratios between certain variables are to be enforced.

*general_cyc *S E Tdt* $t_{per}$ *ctype ncyc nprint nsub*

Use the control statement $S_{ij}d\sigma_j + E_{ij}d\varepsilon_j = T_i dt$ to define cycles. Matrices *S* and *E* and vector *Tdt* are as for *general_inc. Each cycle is divided into *nprint* printing (or plotting) points each calculated using *nsub* substeps. This option allows very versatile control of cycles, including those in which (for instance) some directions are stress-controlled and other strain-controlled, or in which ratios between certain variables are to be enforced.

Cycle type *ctype* may be “saw”, “sine” or “haversine”:




*stress_inc *dt dsig₁ [dsig₂…ndim] nprint nsub*

Stress controlled increment by *dsig*. Print (or plot) data for *nprint* equal steps, each of which is divided into *nsub* calculation substeps.

*strain_inc *dt* $deps_1$ [$deps_{2…ndim}$] *nprint nsub*

Strain controlled increment by *deps.* Print (or plot) data for *nprint* equal steps, each of which is divided into *nsub* calculation substeps.

*stress_targ *dt* $sigtarg_1$ [$sigtarg_{2…ndim}$] *nprint nsub*

Stress control from current stress *sig* to target of *sigtarg.* Print (or plot) data for *nprint* equal steps, each of which is divided into *nsub* calculation substeps.

*strain_targ *dt* $epstarg_1$ [$epstarg_{2…ndim}$] *nprint nsub*

Strain control from current strain *eps* to target of *epstarg.* Print (or plot) data for *nprint* equal steps, each of which is divided into *nsub* calculation substeps.

*stress_cycle $t_{per}$ $sigcyc_1$ [$sigcyc_{2…ndim}$] *ctype ncyc nprint nsub*

Stress controlled cycling *ncyc* times from current stress *sig* to *sig* + *sigcyc* and back to *sig.* Output for each cycle is given for *nprint* equal steps, each of which is divided into *nsub* calculation substeps. Cycle type *ctype* as for *general_cyc.

*strain_cycle $t_{per}$ $epscyc_1$ [$epscyc_{2…ndim}$] *ctype ncyc nprint nsub*

Strain controlled cycling *ncyc* times from current strain *eps* to *eps* + *epscyc* and back to *eps.* Output for each cycle is given for *nprint* equal steps, each of which is divided into *nsub* calculation substeps. Cycle type *ctype* as for *general_cyc.

*strain_test *filename nsub*

Read data from file "*filename*.csv" (lines $eps_1$ [$eps_{2…ndim}$] $sig_1$ [$sig_{2…ndim}$]) and treat the strain data as input. The ".csv" extension is assumed if not given.

*nsub* calculation substeps are used for each strain increment (*i.e.* each line in the input file).

*stress_test *filename nsub*

Read data from file "*filename*.csv" (lines $eps_1$ [$eps_{2…ndim}$] $sig_1$ [$sig_{2…ndim}$]) and treat the stress data as input. The ".csv" extension is assumed if not given.

*nsub* calculation substeps are used for each strain increment (*i.e.* each line in the input file).

*strain_path *filename nsub*

Read data from file "*filename*.csv" (lines $eps_1$ [$eps_{2…ndim}$]) and use the strain data as input strain path. The ".csv" extension is assumed if not given.

*nsub* calculation substeps are used for each strain increment (*i.e.* each line in the input file).

*stress_path *filename nsub*

Read data from file "*filename*.csv" (lines $sig_1$ [$sig_{2…ndim}$]) and use the strain data as input stress path. The ".csv" extension is assumed if not given.

*nsub* calculation substeps are used for each stress increment (*i.e.* each line in the input file).

*start_history

Start recording a history of strain and/or stress changes (specified by *stress_inc *etc*.) which can later be repeated using *run_history as a shorthand for the entire sequence.

*end_history

End recording of history.

*run_history *[N]*

Run a previously recorded history. If the optional parameter *N* is included the history will be run *N* times.

## 4.5 Control of plotted and printed output

*plot [*plotfile*]

Plot the results. If *plotfile* is specified the plot will also be output to "*plotfile*.png" (the .png extension is assumed if not provided). Otherwise the plot is output to "hyper_*model*.png".

*graph *xaxis yaxis* [*plotfile*]

Plot a graph, where *xaxis* and *yaxis* are the specified variable names. For example "*graph t sig" will plot (for a mode 0 model with default names) stress against time. If *plotfile* is specified the plot will also be output to "*plotfile*.png" (the .png extension is assumed if not provided). Otherwise the plot is output to "hyper_*model*.png".

*specialplot [*plotfile*]

Plot the results using special plotting format for the particular model. If *plotfile* is specified the plot will also be output to "*plotfile*.png" (the .png extension is assumed if not provided). Otherwise the plot is output to "hyper_*model*.png".

*colour *col*

Set colour for subsequent plotted curves, *col* values:

| | |
|---|---|
| r | red |
| g | green |
| b | blue (default) |
| c | cyan |
| m | magenta |
| y | yellow |
| k | black |
| w | white (not much use if background is white) |

See https://matplotlib.org/3.1.0/gallery/color/named_colors.html for many other available colours.

*high

Start highlighting within plots (must be matched with *unhigh). May be used several times in one run.

*unhigh

Stop highlighting within plots

*stoprec

Stop recording of stress-strain data (useful to shorten very long plots and files when there are many cycles).

*rec

Restart recording of stress-strain data (the default state).

*printrec [*printfile*]

Print the strains and stresses. If *printfile* is specified the data will also be output to "*printfile*.csv" (the .csv extension is assumed if not provided). Otherwise the data is output to "hyper_*model*.csv".

*pause

Pause output. Hit return to continue.

## 4.6 Example input file

An example input file that illustrates a number of the above features is given below:

| File entry | Comment |
|---|---|
| `*title Hyperplasticity test run` | Title |
| `*mode 0` | Set mode |
| `*model h1epk` | Choose model |
| `*const 200.0 1.2 25.0` | Constants override model default values |
| `*start` | Start test |
| `*strain_inc 1.0 0.04 200 10` | Increment strain by 0.4 |
| `*stress_targ 1.0 0.0 100 10` | Unload to zero stress |
| `*strain_targ 1.0 0.05 100 10` | Strain to 0.5 |
| `#highlight this section` | A comment – will be ignored |
| `*high` | Set highlighting |
| `*stress_inc 1.0 -1.5 150 10` | Unload by stress increment -1.5 |
| `*unhigh` | Unset highlighting |
| | Blank line will be ignored |
| `*stress_cyc 1.0 1.2 saw 5 120 10` | 5 “sawtooth” stress cycles of amplitude 1.2 |
| `*plot` | Plot the results |
| `*end` | Finish |

The figure plotted at the end of the test is shown in Figure 1.

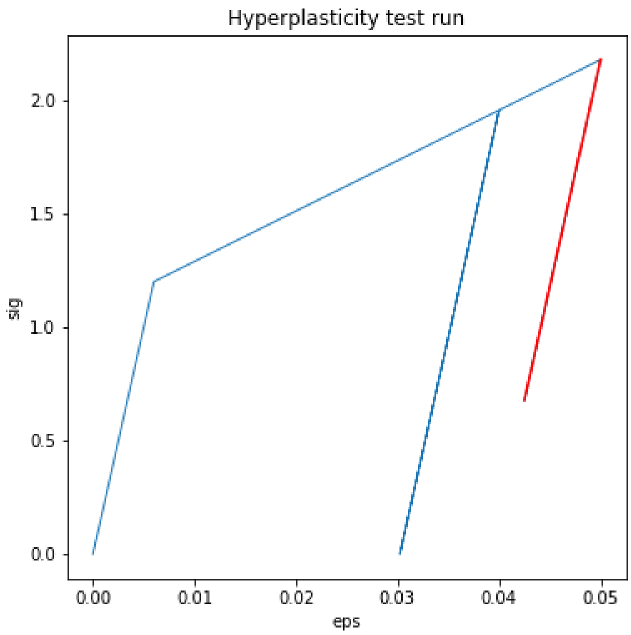



*Figure 1: output figure from example data, note the highlighted section. The cyclic loading at the end of the test is simply elastic*

## 4.7 Running HyperDrive directly from Python without an input file

HyperDrive may also be run by directly invoking a series of commands from Python, in which case an input data file is not needed. The commands are exactly as defined above (without the leading “*”) and the arguments (if required) are provided in the form of a list. The necessary commands must be imported from “HyperDrive.py”, using for instance the form of code below.

Thus exactly the same result as described using the input file given above can be achieved with the following Python code:

```
from HyperDrive import startup
from HyperDrive import Commands as H
startup()
```

```
H.title(["Hyperplasticity test run"]
H.mode([0])
H.model(["h1epk"])
H.const([200.0, 1.2, 25.0])
H.start()
H.strain_inc([1.0, 0.04, 200, 10])
H.stress_targ([1.0, 0.0, 100, 10])
H.strain_targ([1.0, 0.05, 100, 10])
H.high()
H.stress_inc([1.0, -1.5, 150, 10])
H.unhigh()
H.stress_cyc(1.0, 1.2, "saw", 1.2, 5.0, 120, 10])
H.plot()
H.end()
```

# 5 Some examples

This section presents some example models, starting with an extremely simple model and building up to more complex cases.

# 6 Models available

Models currently available are outlined here. Table 2 (at end of document) gives information on the current status of each model.

## 6.1 Mode 0 models

### 6.1.1 h1e – Elastic

Note – because of the way HyperDrive is structured, a yield function must always be specified. However, if only elastic response is required, then the function can be set always to return a value less than zero, so that the plasticity routines are never invoked.

$$f = \frac{E\varepsilon^2}{2}$$

$$g = -\frac{\sigma^2}{2E}$$

| Constant | $E$ |
|---|---|
| Default | 100.0 |

### 6.1.2 h1ep – Elastic perfectly plastic

$$f = \frac{E(\varepsilon - \alpha)^2}{2}$$

$$g = -\frac{\sigma^2}{2E} - \sigma\alpha$$

$$y = |\chi| - k$$

| Constant | $E$ | $k$ |
|---|---|---|
| Default | 100.0 | 1.0 |

### 6.1.3 h1epi – Elastic isotropic hardening plastic

$$f = \frac{E(\varepsilon - \alpha)^2}{2} + \frac{H\alpha^2}{2}$$

$$g = -\frac{\sigma^2}{2E} - \sigma\alpha + \frac{H\alpha^2}{2}$$

$$y = |\chi| - (k_0 + k_1\beta) + \chi_\beta$$

Note – uses second internal variable for hardening parameter $\beta$.

| Constant | $E$ | $k_0$ | $k_1$ |
|---|---|---|---|
| Default | 100.0 | 1.0 | 5.0 |

### 6.1.4 h1epk – Elastic kinematic hardening plastic

$$f = \frac{E(\varepsilon - \alpha)^2}{2} + \frac{H\alpha^2}{2}$$

$$g = -\frac{\sigma^2}{2E} - \sigma\alpha + \frac{H\alpha^2}{2}$$

$$y = |\chi| - k$$

$$w = \frac{\langle |\chi| - k \rangle^2}{2\mu}$$

Note - uses the series implementation.

| Constant | $E$ | $k$ | $H$ |
|---|---|---|---|
| Default | 100.0 | 1.0 | 5.0 |

### 6.1.5 h1epmk_ser – Elastic multisurface kinematic hardening plastic (series)

$$f = \frac{E}{2}\left(\varepsilon - \sum_{n=1}^{N} \alpha_n\right)^2 + \sum_{n=1}^{N} \frac{H_n \alpha_n^2}{2}$$

$$g = -\frac{\sigma^2}{2E} - \sigma \sum_{n=1}^{N} \alpha_n + \sum_{n=1}^{N} \frac{H_n \alpha_n^2}{2}$$

$$y^n = |\chi_n| - k_n, n = 1 \ldots N$$

| Constant | $N$ | $E$ | $\{k_n, n = 1 \ldots N\}$ | $\{H_n\ n = 1 \ldots N\}$ |
|---|---|---|---|---|
| Default | 4 | 100.0 | 0.1, 0.3, 0.6, 1.0 | 100.0, 33.33, 20.0, 10.0 |

### 6.1.6 h1epmk_ser_b – Elastic multi-kin hardening plastic (series, bounding variant)

$$f = \frac{E}{2}\left(\varepsilon - \sum_{n=1}^{N} \alpha_n\right)^2 + \sum_{n=1}^{N} \frac{H_n \alpha_n^2}{2}$$

$$g=-\frac{\sigma^2}{2E}-\sigma\sum_{n=1}^{N}\alpha_n+\sum_{n=1}^{N}\frac{H_n\alpha_n^2}{2}$$

$$y=\sqrt{\sum_{n=1}^{N}\frac{\chi_n^2}{k_n^2}}-1$$

| Constant | $N$ | $E$ | $\{k_n, n = 1…N\}$ | $\{H_n\ n = 1…N\}$ |
|---|---|---|---|---|
| Default | 4 | 100.0 | 0.165, 0.432, 1.613, 1.140 | 100.0, 33.33, 20.0, 10.0 |

### 6.1.7 h1epmk_ser_h – Elastic multi-kin hardening plastic (series, HARM variant)

$$f=\frac{E}{2}\left(\varepsilon-\sum_{n=1}^{N}\alpha_n-\alpha_r\right)^2+\sum_{n=1}^{N}\frac{H_n\alpha_n^2}{2}$$

$$g=-\frac{\sigma^2}{2E}-\sigma\sum_{n=1}^{N}\alpha_n-\sigma\alpha_r+\sum_{n=1}^{N}\frac{H_n\alpha_n^2}{2}$$

$$y^n=\frac{|\chi_n|}{k_n}-1.0+\frac{R}{k_{ref}}\left(|\chi_r|-|\sigma|\right), n=1…N$$

$$\dot{\alpha}_n=\frac{\Lambda^n}{k_n}S(\chi_n), n=1…N\,,\ |\dot{\alpha}_n|=\frac{\Lambda^n}{k_n}, n=1…N$$

$$\dot{\alpha}_r=\frac{R}{k_{ref}}S(\chi_r)\sum_{n=1}^{N}\Lambda^n=\frac{R}{k_{ref}}S(\sigma)\sum_{n=1}^{N}k_n|\dot{\alpha}_n|$$

Note – the ratcheting strain is included as the $(N+1)^{th}$ internal variable.

| Constant | $N$ | $E$ | $\{k_n, n = 1…N\}$ | $\{H_n\ n = 1…N\}$ | $R/k_{ref}$ |
|---|---|---|---|---|---|
| Default | 4 | 100.0 | 0.103, 0.324, 0.645, 1.053 | 100.0, 33.33, 20.0, 10.0 | 0.1 |

### 6.1.8 h1epmk_par – Elastic multisurface kinematic hardening plastic (parallel)

$$f=\frac{E_{\text{inf}}}{2}\varepsilon^2+\sum_{n=1}^{N}\frac{H_n(\varepsilon-\alpha_n)^2}{2}$$

$$g=-\frac{\sigma^2}{2E}-\sigma\sum_{n=1}^{N}\alpha_n+\sum_{n=1}^{N}\frac{H_n\alpha_n^2}{2}$$

$$y^n=|\chi_n|-k_n, n=1…N$$

| Constant | $N$ | $E_{inf}$ | $\{k_n, n = 1…N\}$ | $\{H_n\ n = 1…N\}$ |
|---|---|---|---|---|
| Default | 4 | 100.0 | 1.0 | 10.0 |

### 6.1.9 h1epmk_par_b – Elastic multi-kin hardening plastic (parallel, bounding variant)

$$f = \frac{E}{2}\left(\varepsilon - \sum_{n=1}^{N} \alpha_n\right)^2 + \sum_{n=1}^{N} \frac{H_n \alpha_n^2}{2}$$

$$g = -\frac{\sigma^2}{2E} - \sigma \sum_{n=1}^{N} \alpha_n + \sum_{n=1}^{N} \frac{H_n \alpha_n^2}{2}$$

$$y = \sqrt{\sum_{n=1}^{N} \frac{\chi_n^2}{k_n^2}} - 1$$

| Constant | $N$ | $E$ | $\{k_n, n = 1 \ldots N\}$ | $\{H_n\ n = 1 \ldots N\}$ |
|---|---|---|---|---|
| Default | 4 | 100.0 | 1.0 | 10.0 |

### 6.1.10 h1epmk_par_h – Elastic multi-kin hardening plastic (parallel, HARM variant)

$$f = \frac{E}{2}\left(\varepsilon - \sum_{n=1}^{N} \alpha_n\right)^2 + \sum_{n=1}^{N} \frac{H_n \alpha_n^2}{2}$$

$$g = -\frac{\sigma^2}{2E} - \sigma \sum_{n=1}^{N} \alpha_n + \sum_{n=1}^{N} \frac{H_n \alpha_n^2}{2}$$

$$y = \sqrt{\sum_{n=1}^{N} \frac{\chi_n^2}{k_n^2}} - 1$$

Note – the ratcheting strain is included as the $(N+1)^{\text{th}}$ internal variable.

| Constant | $N$ | $E$ | $\{k_n, n = 1 \ldots N\}$ | $\{H_n\ n = 1 \ldots N\}$ | $R$ |
|---|---|---|---|---|---|
| Default | 4 | 100.0 | 1.0 | 10.0 | 0.1 |

### 6.1.11 h1epmk_nest – Elastic multisurface kinematic hardening plastic (nested)

$$f = \frac{E}{2}\left(\varepsilon - \sum_{n=1}^{N} \alpha_n\right)^2 + \sum_{n=1}^{N} \frac{H_n \alpha_n^2}{2}$$

$$g = -\frac{\sigma^2}{2E} - \sigma \sum_{n=1}^{N} \alpha_n + \sum_{n=1}^{N} \frac{H_n \alpha_n^2}{2}$$

$$y^n = \left|\chi_n\right| - k_n, n = 1 \ldots N$$

| Constant | $N$ | $E$ | $\{k_n, n = 1 \ldots N\}$ | $\{H_n\ n = 1 \ldots N\}$ |
|---|---|---|---|---|
| Default | 4 | 100.0 | 1.0 | 10.0 |

### 6.1.12 h1epmk_nest_b – Elastic multi-kin hardening plastic (nested, bounding variant)

$$f = \frac{E}{2}\left(\varepsilon - \sum_{n=1}^{N} \alpha_n\right)^2 + \sum_{n=1}^{N} \frac{H_n \alpha_n^2}{2}$$

$$g = -\frac{\sigma^2}{2E} - \sigma\sum_{n=1}^{N}\alpha_n + \sum_{n=1}^{N}\frac{H_n\alpha_n^2}{2}$$

$$y = \sqrt{\sum_{n=1}^{N}\frac{\chi_n^2}{k_n^2}} - 1$$

| Constant | $N$ | $E$ | $\{k_n, n = 1 \ldots N\}$ | $\{H_n\ n = 1 \ldots N\}$ |
|---|---|---|---|---|
| Default | 4 | 100.0 | 1.0 | 10.0 |

### 6.1.13 h1epmk_nest_h – Elastic multi-kin hardening plastic (nested, HARM variant)

$$f = \frac{E}{2}\left(\varepsilon - \sum_{n=1}^{N}\alpha_n\right)^2 + \sum_{n=1}^{N}\frac{H_n\alpha_n^2}{2}$$

$$g = -\frac{\sigma^2}{2E} - \sigma\sum_{n=1}^{N}\alpha_n + \sum_{n=1}^{N}\frac{H_n\alpha_n^2}{2}$$

$$y = \sqrt{\sum_{n=1}^{N}\frac{\chi_n^2}{k_n^2}} - 1$$

Note – the ratcheting strain is included as the $(N+1)^{\text{th}}$ internal variable.

| Constant | $N$ | $E$ | $\{k_n, n = 1 \ldots N\}$ | $\{H_n\ n = 1 \ldots N\}$ | $R$ |
|---|---|---|---|---|---|
| Default | 4 | 100.0 | 1.0 | 10.0 | 0.1 |

## 6.2 Mode 1 models

### 6.2.1 hnepmk_ser – Elastic multisurface kinematic hardening plastic (series)

With summation over dimension *i* (which is of dimension *ndim*):

$$f = \frac{E}{2}\left(\varepsilon_i - \sum_{n=1}^{N}\alpha_{ni}\right)\left(\varepsilon_i - \sum_{n=1}^{N}\alpha_{ni}\right) + \sum_{n=1}^{N}\frac{H_n\alpha_{ni}\alpha_{ni}}{2}$$

$$g = -\frac{\sigma_i\sigma_i}{2E} - \sigma_i\sum_{n=1}^{N}\alpha_{ni} + \sum_{n=1}^{N}\frac{H_n\alpha_{ni}\alpha_{ni}}{2}$$

$$y^n = \frac{\chi_{ni}\chi_{ni} - k_n^2}{2}, n = 1 \ldots N$$

Constants required:

*ndim*, (+ parameters as for h1epmk_ser)

### 6.2.2 hnepmk_ser_b – Elastic multi-kin hardening plastic (series, bounding surface variant)

With summation over dimension *i* (which is of dimension *ndim*):

$$f = \frac{E}{2}\left(\varepsilon_i - \sum_{n=1}^{N}\alpha_{ni}\right)\left(\varepsilon_i - \sum_{n=1}^{N}\alpha_{ni}\right) + \sum_{n=1}^{N}\frac{H_n\alpha_{ni}\alpha_{ni}}{2}$$

$$g = -\frac{\sigma_i \sigma_i}{2E} - \sigma_i \sum_{n=1}^{N} \alpha_{ni} + \sum_{n=1}^{N} \frac{H_n \alpha_{ni} \alpha_{ni}}{2}$$

$$y = \frac{1}{2}\left(\left(\sum_{n=1}^{N} \frac{\chi_{ni} \chi_{ni}}{k_n^2}\right) - 1\right)$$

Constants required:

*ndim*, (+ parameters as for h1epmk_ser_b)

### 6.2.3 hfrict – Simple frictional model

$$f = \frac{K}{2}(\varepsilon_v - \alpha_v)^2 + \frac{3G}{2}(\varepsilon_s - \alpha_s)^2$$

$$g = -\frac{\sigma_p^2}{2K} - \frac{\sigma_q^2}{2 \times 3G} - \sigma_p \alpha_v - \sigma_q \alpha_s$$

$$y = |\chi_q| - N\chi_p - M\sigma_p$$

| Constant | $K$ | $G$ | $M$ | $N$ |
|---|---|---|---|---|
| Default | 100.0 | 80.0 | 1.0 | 0.3 |

### 6.2.4 hmcc – Modified Cam Clay

$$f = p_r \kappa^* \exp\left(\frac{\varepsilon_v - \alpha_v}{\kappa^*}\right) + \frac{3G}{2}(\varepsilon_s - \alpha_s)^2$$

$$g = -p_r \kappa^* \operatorname{ilog}\left(\frac{\sigma_p}{p_r}\right) - \frac{\sigma_q^2}{2 \times 3G} - \sigma_p \alpha_v - \sigma_q \alpha_s$$

where $\operatorname{ilog}(x) = x\log(x) - x$

$$y = \chi_p^2 + \frac{\chi_q^2}{M^2} - \chi_p p_c$$

| Constant | $p_r$ | $\lambda^*$ | $\kappa^*$ | $M$ | $G$ | $p_{co}$ |
|---|---|---|---|---|---|---|
| Default | 100.0 | 0.2 | 0.05 | 1.0 | 200.0 | 20.0 |

## 6.3 Mode 2 Models

None yet implemented.

# 7 Code checking

A utility routine "check" is provided that checks the differentials of the basic functions against numerically derived values. If this is used it is recommended (for ease of operation) that suitable values of *check_eps*, *check_sig*, *check_alp* and *check_chi* be given in the relevant model file. These are then used to define parameter sets at which the numerical checks are made. The routine currently carries out 32 checks on model consistency.

After loading the HyperDrive routines, the check routine is invoked by typing "`check()`" or "`check("`*model_file*`")`".

The routine carries out a number of basic consistency checks, and (where possible) compares the results from any user supplied differentials, the automatic differentials and the numerical ones. Checks on dimensional consistency of output are also made. Occasionally a check will "fail" even if the comparison is satisfactory, as tolerances may be too tight.

In general, however, after writing any new user-defined differentials it is advised that the check() routine is run. It is unlikely that faulty code would manage to pass all the tests.

# 8 Notes on incremental forms: rate independent

## 8.1 Methods based on *f*-*y* functions

### 8.1.1 Basic functions and their derivatives

$$f\left(\varepsilon_i, \alpha_j^m\right)$$

$$y^p\left(\chi_i^m, \varepsilon_j, \alpha_k^n\right)$$

Derivatives

$$\sigma_i = \frac{\partial f}{\partial \varepsilon_i}$$

$$\chi_i^m = -\frac{\partial f}{\partial \alpha_i^m}$$

Increments

$$d\sigma_i = \frac{\partial^2 f}{\partial \varepsilon_i \partial \varepsilon_j} d\varepsilon_j + \frac{\partial^2 f}{\partial \varepsilon_i \partial \alpha_j^m} d\alpha_j^m$$

$$d\chi_i^m = -\frac{\partial^2 f}{\partial \alpha_i^m \partial \varepsilon_j} d\varepsilon_j - \frac{\partial^2 f}{\partial \alpha_i^m \partial \alpha_j^n} d\alpha_j^n$$

Consistency condition and flow rule during yield

$$dy^p = -ay_o^p = \frac{\partial y^p}{\partial \chi_i^m} d\chi_i^m + \frac{\partial y^p}{\partial \varepsilon_i} d\varepsilon_i + \frac{\partial y^p}{\partial \alpha_i^m} d\alpha_i^m$$

$$d\alpha_i^m = \Lambda^p \frac{\partial y^p}{\partial \chi_i^m}$$

### 8.1.2 Development for strain control

Assume $d\varepsilon_i$ specified.

Substitute for generalized stress increment in consistency condition

$$-ay_o^p = \frac{\partial y^p}{\partial \chi_i^m}\left(-\frac{\partial^2 f}{\partial \alpha_i^m \partial \varepsilon_j} d\varepsilon_j - \frac{\partial^2 f}{\partial \alpha_i^m \partial \alpha_j^n} d\alpha_j^n\right) + \frac{\partial y^p}{\partial \varepsilon_i} d\varepsilon_i + \frac{\partial y^p}{\partial \alpha_i^m} d\alpha_i^m$$

$$= \left(\frac{\partial y^p}{\partial \varepsilon_j} - \frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 f}{\partial \alpha_i^m \partial \varepsilon_j}\right) d\varepsilon_j + \left(\frac{\partial y^p}{\partial \alpha_j^n} - \frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 f}{\partial \alpha_i^m \partial \alpha_j^n}\right) d\alpha_j^n$$

$$= \left(\frac{\partial y^p}{\partial \varepsilon_j} - \frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 f}{\partial \alpha_i^m \partial \varepsilon_j}\right) d\varepsilon_j + \left(\frac{\partial y^p}{\partial \alpha_j^n} - \frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 f}{\partial \alpha_i^m \partial \alpha_j^n}\right) \Lambda^q \frac{\partial y^q}{\partial \chi_j^n}$$

Re-arrange to solve for plastic multipliers as function of strain increment

$$-\left(\frac{\partial y^p}{\partial \alpha_j^n} - \frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 f}{\partial \alpha_i^m \partial \alpha_j^n}\right)\frac{\partial y^q}{\partial \chi_j^n}\Lambda^q = ay_o^p + \left(\frac{\partial y^p}{\partial \varepsilon_j} - \frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 f}{\partial \alpha_i^m \partial \varepsilon_j}\right) d\varepsilon_j$$

### 8.1.3 Development for stress control

Assume $d\sigma_i$ specified.

Obtain compliance matrix

$$C_{ij} = \left(\frac{\partial^2 f}{\partial \varepsilon_i \partial \varepsilon_j}\right)^{-1}$$

Re-arrange increment and substitute compliance matrix

$$\frac{\partial^2 f}{\partial \varepsilon_i \partial \varepsilon_j} d\varepsilon_j = d\sigma_i - \frac{\partial^2 f}{\partial \varepsilon_i \partial \alpha_j^m} d\alpha_j^m$$

$$C_{ki}\frac{\partial^2 f}{\partial \varepsilon_i \partial \varepsilon_j} d\varepsilon_j = d\varepsilon_k = C_{ki}\left(d\sigma_i - \frac{\partial^2 f}{\partial \varepsilon_i \partial \alpha_j^m} d\alpha_j^m\right)$$

Substitute for generalized stress increment in consistency condition

$$-ay_o^p = \frac{\partial y^p}{\partial \chi_i^m}\left(-\frac{\partial^2 f}{\partial \alpha_i^m \partial \varepsilon_j} d\varepsilon_j - \frac{\partial^2 f}{\partial \alpha_i^m \partial \alpha_j^n} d\alpha_j^n\right) + \frac{\partial y^p}{\partial \varepsilon_i} d\varepsilon_i + \frac{\partial y^p}{\partial \alpha_i^m} d\alpha_i^m$$

$$= \left(\frac{\partial y^p}{\partial \varepsilon_j} - \frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 f}{\partial \alpha_i^m \partial \varepsilon_j}\right) C_{jk}\left(d\sigma_k - \frac{\partial^2 f}{\partial \varepsilon_k \partial \alpha_l^n} d\alpha_l^n\right) + \left(\frac{\partial y^p}{\partial \alpha_j^n} - \frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 f}{\partial \alpha_i^m \partial \alpha_j^n}\right) d\alpha_j^n$$

$$= \left(\frac{\partial y^p}{\partial \varepsilon_j} - \frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 f}{\partial \alpha_i^m \partial \varepsilon_j}\right) C_{jk} d\sigma_k +$$

$$\left(-\left(\frac{\partial y^p}{\partial \varepsilon_j} - \frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 f}{\partial \alpha_i^m \partial \varepsilon_j}\right) C_{jk}\frac{\partial^2 f}{\partial \varepsilon_k \partial \alpha_l^n} + \left(\frac{\partial y^p}{\partial \alpha_l^n} - \frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 f}{\partial \alpha_i^m \partial \alpha_l^n}\right)\right) d\alpha_l^n$$

$$= \left(\frac{\partial y^p}{\partial \varepsilon_j} - \frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 f}{\partial \alpha_i^m \partial \varepsilon_j}\right) C_{jk} d\sigma_k +$$

$$\left(-\left(\frac{\partial y^p}{\partial \varepsilon_j} - \frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 f}{\partial \alpha_i^m \partial \varepsilon_j}\right) C_{jk}\frac{\partial^2 f}{\partial \varepsilon_k \partial \alpha_l^n} + \left(\frac{\partial y^p}{\partial \alpha_l^n} - \frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 f}{\partial \alpha_i^m \partial \alpha_l^n}\right)\right) \Lambda^q \frac{\partial y^q}{\partial \chi_l^n}$$

Re-arrange to solve for plastic multipliers as function of strain increment

$$\left(\left(\frac{\partial y^p}{\partial \varepsilon_j} - \frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 f}{\partial \alpha_i^m \partial \varepsilon_j}\right) C_{jk}\frac{\partial^2 f}{\partial \varepsilon_k \partial \alpha_l^n} - \left(\frac{\partial y^p}{\partial \alpha_l^n} - \frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 f}{\partial \alpha_i^m \partial \alpha_l^n}\right)\right)\frac{\partial y^q}{\partial \chi_l^n}\Lambda^q =$$

$$ay_o^p + \left(\frac{\partial y^p}{\partial \varepsilon_j} - \frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 f}{\partial \alpha_i^m \partial \varepsilon_j}\right) C_{jk} d\sigma_k$$

### 8.1.4 Development for general control

Assume control statement of the form:

$$S_{ij} d\sigma_j + E_{ij} d\varepsilon_j = T_i dt$$

Re-arrange control statement

$$S_{ij}\left(\frac{\partial^2 f}{\partial \varepsilon_j \partial \varepsilon_k} d\varepsilon_k + \frac{\partial^2 f}{\partial \varepsilon_j \partial \alpha_k^m} d\alpha_k^m\right) + E_{ik} d\varepsilon_k = T_i dt$$

$$\left(E_{ik} + S_{ij}\frac{\partial^2 f}{\partial \varepsilon_j \partial \varepsilon_k}\right) d\varepsilon_k = T_i dt - S_{ij}\frac{\partial^2 f}{\partial \varepsilon_j \partial \alpha_k^m} d\alpha_k^m$$

Derive modified control statement

$$P_{ik} = \left(E_{ik} + S_{ij}\frac{\partial^2 f}{\partial \varepsilon_j \partial \varepsilon_k}\right)^{-1}$$

$$P_{li}\left(E_{ik}+S_{ij}\frac{\partial^2 f}{\partial\varepsilon_j\partial\varepsilon_k}\right)d\varepsilon_k = d\varepsilon_l = P_{li}\left(T_i dt - S_{ij}\frac{\partial^2 f}{\partial\varepsilon_j\partial\alpha_k^m}d\alpha_k^m\right)$$

Substitute for generalized stress increment in consistency condition

$$\begin{aligned}
-ay_o^p &= \frac{\partial y^p}{\partial\chi_i^m}\left(-\frac{\partial^2 f}{\partial\alpha_i^m\partial\varepsilon_j}d\varepsilon_j - \frac{\partial^2 f}{\partial\alpha_i^m\partial\alpha_j^n}d\alpha_j^n\right)+\frac{\partial y^p}{\partial\varepsilon_j}d\varepsilon_j+\frac{\partial y^p}{\partial\alpha_i^m}d\alpha_i^m \\
&= \left(\frac{\partial y^p}{\partial\varepsilon_j}-\frac{\partial y^p}{\partial\chi_i^m}\frac{\partial^2 f}{\partial\alpha_i^m\partial\varepsilon_j}\right)d\varepsilon_j+\left(\frac{\partial y^p}{\partial\alpha_j^n}-\frac{\partial y^p}{\partial\chi_i^m}\frac{\partial^2 f}{\partial\alpha_i^m\partial\alpha_j^n}\right)d\alpha_j^n \\
&= \left(\frac{\partial y^p}{\partial\varepsilon_j}-\frac{\partial y^p}{\partial\chi_i^m}\frac{\partial^2 f}{\partial\alpha_i^m\partial\varepsilon_j}\right)P_{jr}\left(T_r dt - S_{rl}\frac{\partial^2 f}{\partial\varepsilon_l\partial\alpha_k^m}d\alpha_k^m\right)+\left(\frac{\partial y^p}{\partial\alpha_j^n}-\frac{\partial y^p}{\partial\chi_i^m}\frac{\partial^2 f}{\partial\alpha_i^m\partial\alpha_j^n}\right)d\alpha_j^n \\
&= \left(\frac{\partial y^p}{\partial\varepsilon_j}-\frac{\partial y^p}{\partial\chi_i^m}\frac{\partial^2 f}{\partial\alpha_i^m\partial\varepsilon_j}\right)P_{jr}T_r dt + \\
&\quad \left(-\left(\frac{\partial y^p}{\partial\varepsilon_j}-\frac{\partial y^p}{\partial\chi_i^m}\frac{\partial^2 f}{\partial\alpha_i^m\partial\varepsilon_j}\right)P_{jr}S_{rl}\frac{\partial^2 f}{\partial\varepsilon_l\partial\alpha_k^n}+\left(\frac{\partial y^p}{\partial\alpha_k^n}-\frac{\partial y^p}{\partial\chi_i^m}\frac{\partial^2 f}{\partial\alpha_i^m\partial\alpha_k^n}\right)\right)d\alpha_k^n
\end{aligned}$$

Re-arrange to solve for plastic multipliers

$$\left(\left(\frac{\partial y^p}{\partial\varepsilon_j}-\frac{\partial y^p}{\partial\chi_i^m}\frac{\partial^2 f}{\partial\alpha_i^m\partial\varepsilon_j}\right)P_{jr}S_{rl}\frac{\partial^2 f}{\partial\varepsilon_l\partial\alpha_k^n}-\left(\frac{\partial y^p}{\partial\alpha_k^n}-\frac{\partial y^p}{\partial\chi_i^m}\frac{\partial^2 f}{\partial\alpha_i^m\partial\alpha_k^n}\right)\right)\frac{\partial y^q}{\partial\chi_k^n}\Lambda^q =$$

$$ay_o^p+\left(\frac{\partial y^p}{\partial\varepsilon_j}-\frac{\partial y^p}{\partial\chi_i^m}\frac{\partial^2 f}{\partial\alpha_i^m\partial\varepsilon_j}\right)P_{jr}T_r dt$$

## 8.2 Methods based on *g*-*y* functions

### 8.2.1 Basic functions and their derivatives

$$g\left(\sigma_i,\alpha_j^m\right)$$

$$y^p\left(\chi_i^m,\sigma_j,\alpha_k^n\right)$$

Derivatives

$$\varepsilon_i = -\frac{\partial g}{\partial\sigma_i}$$

$$\chi_i^m = -\frac{\partial g}{\partial\alpha_i^m}$$

Increments

$$d\varepsilon_i = -\frac{\partial^2 g}{\partial\sigma_i\partial\sigma_j}d\sigma_j - \frac{\partial^2 g}{\partial\sigma_i\partial\alpha_j^m}d\alpha_j^m$$

$$d\chi_i^m = -\frac{\partial^2 g}{\partial\alpha_i^m \partial\sigma_j} d\sigma_j - \frac{\partial^2 g}{\partial\alpha_i^m \partial\alpha_j^n} d\alpha_j^n$$

Consistency condition and flow rule during yield

$$dy^p = -ay_o^p = \frac{\partial y^p}{\partial\chi_i^m} d\chi_i^m + \frac{\partial y^p}{\partial\sigma_i} d\sigma_i + \frac{\partial y^p}{\partial\alpha_i^m} d\alpha_i^m$$

$$d\alpha_i^m = \Lambda^p \frac{\partial y^p}{\partial\chi_i^m}$$

### 8.2.2 Development for stress control

Assume $d\sigma_i$ specified.

Substitute for generalized stress increment in consistency condition

$$\begin{aligned}
-ay_o^p &= \frac{\partial y^p}{\partial\chi_i^m}\left(-\frac{\partial^2 g}{\partial\alpha_i^m \partial\sigma_j} d\sigma_j - \frac{\partial^2 g}{\partial\alpha_i^m \partial\alpha_j^n} d\alpha_j^n\right) + \frac{\partial y^p}{\partial\sigma_i} d\sigma_i + \frac{\partial y^p}{\partial\alpha_i^m} d\alpha_i^m \\
&= \left(\frac{\partial y^p}{\partial\sigma_j} - \frac{\partial y^p}{\partial\chi_i^m}\frac{\partial^2 g}{\partial\alpha_i^m \partial\sigma_j}\right) d\sigma_j + \left(\frac{\partial y^p}{\partial\alpha_j^n} - \frac{\partial y^p}{\partial\chi_i^m}\frac{\partial^2 g}{\partial\alpha_i^m \partial\alpha_j^n}\right) d\alpha_j^n \\
&= \left(\frac{\partial y^p}{\partial\sigma_j} - \frac{\partial y^p}{\partial\chi_i^m}\frac{\partial^2 g}{\partial\alpha_i^m \partial\sigma_j}\right) d\sigma_j + \left(\frac{\partial y^p}{\partial\alpha_j^n} - \frac{\partial y^p}{\partial\chi_i^m}\frac{\partial^2 g}{\partial\alpha_i^m \partial\alpha_j^n}\right) \Lambda^q \frac{\partial y^q}{\partial\chi_j^n}
\end{aligned}$$

Re-arrange to solve for plastic multipliers as function of stress increment

$$-\left(\frac{\partial y^p}{\partial\alpha_j^n} - \frac{\partial y^p}{\partial\chi_i^m}\frac{\partial^2 g}{\partial\alpha_i^m \partial\alpha_j^n}\right)\frac{\partial y^q}{\partial\chi_j^n}\Lambda^q = ay_o^p\left(\frac{\partial y^p}{\partial\sigma_j} - \frac{\partial y^p}{\partial\chi_i^m}\frac{\partial^2 g}{\partial\alpha_i^m \partial\sigma_j}\right) d\sigma_j$$

### 8.2.3 Development for strain control

Assume $d\varepsilon_i$ specified.

Obtain stiffness matrix

$$D_{ij} = \left(-\frac{\partial^2 g}{\partial\sigma_i \partial\sigma_j}\right)^{-1}$$

Re-arrange increment and substitute stiffness matrix

$$-\frac{\partial^2 g}{\partial\sigma_i \partial\sigma_j} d\sigma_j = d\varepsilon_i + \frac{\partial^2 g}{\partial\sigma_i \partial\alpha_j^m} d\alpha_j^m$$

$$D_{ki}\left(-\frac{\partial^2 g}{\partial\sigma_i \partial\sigma_j}\right) d\sigma_j = d\sigma_k = D_{ki}\left(d\varepsilon_i + \frac{\partial^2 g}{\partial\sigma_i \partial\alpha_j^m} d\alpha_j^m\right)$$

Substitute for generalized stress increment in consistency condition

$$-ay_o^p = \frac{\partial y^p}{\partial \chi_i^m}\left(-\frac{\partial^2 g}{\partial \alpha_i^m \partial \sigma_j} d\sigma_j - \frac{\partial^2 g}{\partial \alpha_i^m \partial \alpha_j^n} d\alpha_j^n\right) + \frac{\partial y^p}{\partial \sigma_i} d\sigma_i + \frac{\partial y^p}{\partial \alpha_i^m} d\alpha_i^m$$

$$= \left(\frac{\partial y^p}{\partial \sigma_j} - \frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 g}{\partial \alpha_i^m \partial \sigma_j}\right) D_{jk}\left(d\varepsilon_k + \frac{\partial^2 g}{\partial \sigma_k \partial \alpha_l^n} d\alpha_l^n\right) + \left(\frac{\partial y^p}{\partial \alpha_j^n} - \frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 g}{\partial \alpha_i^m \partial \alpha_j^n}\right) d\alpha_j^n$$

$$= \left(\frac{\partial y^p}{\partial \sigma_j} - \frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 g}{\partial \alpha_i^m \partial \sigma_j}\right) D_{jk} d\varepsilon_k +$$

$$\left(\left(\frac{\partial y^p}{\partial \sigma_j} - \frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 g}{\partial \alpha_i^m \partial \sigma_j}\right) D_{jk}\frac{\partial^2 g}{\partial \sigma_k \partial \alpha_l^n} + \left(\frac{\partial y^p}{\partial \alpha_l^n} - \frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 g}{\partial \alpha_i^m \partial \alpha_l^n}\right)\right) d\alpha_l^n$$

$$= \left(\frac{\partial y^p}{\partial \sigma_j} - \frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 g}{\partial \alpha_i^m \partial \sigma_j}\right) D_{jk} d\varepsilon_k +$$

$$\left(\left(\frac{\partial y^p}{\partial \sigma_j} - \frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 g}{\partial \alpha_i^m \partial \sigma_j}\right) D_{jk}\frac{\partial^2 g}{\partial \sigma_k \partial \alpha_l^n} + \left(\frac{\partial y^p}{\partial \alpha_l^n} - \frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 g}{\partial \alpha_i^m \partial \alpha_l^n}\right)\right) \Lambda^q \frac{\partial y^q}{\partial \chi_l^n}$$

Re-arrange to solve for plastic multipliers in terms of strain increment

$$\left(-\left(\frac{\partial y^p}{\partial \sigma_j} - \frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 g}{\partial \alpha_i^m \partial \sigma_j}\right) D_{jk}\frac{\partial^2 g}{\partial \sigma_k \partial \alpha_l^n} - \left(\frac{\partial y^p}{\partial \alpha_l^n} - \frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 g}{\partial \alpha_i^m \partial \alpha_l^n}\right)\right)\frac{\partial y^q}{\partial \chi_l^n}\Lambda^q =$$

$$ay_o^p + \left(\frac{\partial y^p}{\partial \sigma_j} - \frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 g}{\partial \alpha_i^m \partial \sigma_j}\right) D_{jk} d\varepsilon_k$$

### 8.2.4 Development for general control

Assume control statement of the form:

$$S_{ij} d\sigma_j + E_{ij} d\varepsilon_j = T_i dt$$

Re-arrange control statement

$$S_{ik} d\sigma_k + E_{ij}\left(-\frac{\partial^2 g}{\partial \sigma_j \partial \sigma_k} d\sigma_k - \frac{\partial^2 g}{\partial \sigma_j \partial \alpha_k^m} d\alpha_k^m\right) = T_i dt$$

$$\left(S_{ik} - E_{ij}\frac{\partial^2 g}{\partial \sigma_j \partial \sigma_k}\right) d\sigma_k = T_i dt + E_{ij}\frac{\partial^2 g}{\partial \sigma_j \partial \alpha_k^m} d\alpha_k^m$$

Derive modified control statement

$$Q_{ik} = \left(S_{ik} - E_{ij}\frac{\partial^2 g}{\partial \sigma_j \partial \sigma_k}\right)^{-1}$$

$$Q_{li}\left(S_{ik}-E_{ij}\frac{\partial^2 g}{\partial\sigma_j\partial\sigma_k}\right)d\sigma_k=d\sigma_l=Q_{li}\left(T_i dt+E_{ij}\frac{\partial^2 g}{\partial\sigma_j\partial\alpha_k^m}d\alpha_k^m\right)$$

Substitute for generalized stress increment in consistency condition

$$\begin{aligned}
-ay_o^p &= \frac{\partial y^p}{\partial\chi_i^m}\left(-\frac{\partial^2 g}{\partial\alpha_i^m\partial\sigma_j}d\sigma_j-\frac{\partial^2 g}{\partial\alpha_i^m\partial\alpha_j^n}d\alpha_j^n\right)+\frac{\partial y^p}{\partial\sigma_j}d\sigma_j+\frac{\partial y^p}{\partial\alpha_i^m}d\alpha_i^m \\
&=\left(\frac{\partial y^p}{\partial\sigma_j}-\frac{\partial y^p}{\partial\chi_i^m}\frac{\partial^2 g}{\partial\alpha_i^m\partial\sigma_j}\right)d\sigma_j+\left(\frac{\partial y^p}{\partial\alpha_j^n}-\frac{\partial y^p}{\partial\chi_i^m}\frac{\partial^2 g}{\partial\alpha_i^m\partial\alpha_j^n}\right)d\alpha_j^n \\
&=\left(\frac{\partial y^p}{\partial\sigma_j}-\frac{\partial y^p}{\partial\chi_i^m}\frac{\partial^2 g}{\partial\alpha_i^m\partial\sigma_j}\right)Q_{jr}\left(T_r dt+E_{rl}\frac{\partial^2 g}{\partial\sigma_l\partial\alpha_k^m}d\alpha_k^m\right)+\left(\frac{\partial y^p}{\partial\alpha_j^n}-\frac{\partial y^p}{\partial\chi_i^m}\frac{\partial^2 g}{\partial\alpha_i^m\partial\alpha_j^n}\right)d\alpha_j^n \\
&=\left(\frac{\partial y^p}{\partial\sigma_j}-\frac{\partial y^p}{\partial\chi_i^m}\frac{\partial^2 g}{\partial\alpha_i^m\partial\sigma_j}\right)Q_{jr}T_r dt+ \\
&\quad\left(\left(\frac{\partial y^p}{\partial\sigma_j}-\frac{\partial y^p}{\partial\chi_i^m}\frac{\partial^2 g}{\partial\alpha_i^m\partial\sigma_j}\right)Q_{jr}E_{rl}\frac{\partial^2 g}{\partial\sigma_l\partial\alpha_k^n}+\left(\frac{\partial y^p}{\partial\alpha_j^n}-\frac{\partial y^p}{\partial\chi_i^m}\frac{\partial^2 g}{\partial\alpha_i^m\partial\alpha_j^n}\right)\right)d\alpha_k^n
\end{aligned}$$

Re-arrange to solve for plastic multipliers

$$\left(-\left(\frac{\partial y^p}{\partial\sigma_j}-\frac{\partial y^p}{\partial\chi_i^m}\frac{\partial^2 g}{\partial\alpha_i^m\partial\sigma_j}\right)Q_{jr}E_{rl}\frac{\partial^2 g}{\partial\sigma_l\partial\alpha_k^n}-\left(\frac{\partial y^p}{\partial\alpha_k^n}-\frac{\partial y^p}{\partial\chi_i^m}\frac{\partial^2 g}{\partial\alpha_i^m\partial\alpha_k^n}\right)\right)\frac{\partial y^q}{\partial\chi_k^n}\Lambda^q=$$

$$ay_o^p+\left(\frac{\partial y^p}{\partial\sigma_j}-\frac{\partial y^p}{\partial\chi_i^m}\frac{\partial^2 g}{\partial\alpha_i^m\partial\sigma_j}\right)Q_{jr}T_r dt$$

# 9 Notes on incremental forms: rate dependent

## 9.1 Methods based on *f*-*w* functions

### 9.1.1 Basic functions and their derivatives

$$f\left(\varepsilon_i,\alpha_j^m\right)$$

$$w\left(\chi_i^m,\varepsilon_j,\alpha_k^n\right)$$

Derivatives

$$\sigma_i=\frac{\partial f}{\partial\varepsilon_i}$$

$$\chi_i^m=-\frac{\partial f}{\partial\alpha_i^m}$$

Increments

$$d\sigma_i = \frac{\partial^2 f}{\partial \varepsilon_i \partial \varepsilon_j} d\varepsilon_j + \frac{\partial^2 f}{\partial \varepsilon_i \partial \alpha_j^m} d\alpha_j^m$$

$$d\chi_i^m = -\frac{\partial^2 f}{\partial \alpha_i^m \partial \varepsilon_j} d\varepsilon_j - \frac{\partial^2 f}{\partial \alpha_i^m \partial \alpha_j^n} d\alpha_j^n$$

$$d\alpha_i^m = \frac{\partial w}{\partial \chi_i^m} dt$$

### 9.1.2 Development for strain control

Assume $d\varepsilon_i$ specified.

$$d\alpha_i^m = \frac{\partial w}{\partial \chi_i^m} dt$$

Then

$$d\sigma_i = \frac{\partial^2 f}{\partial \varepsilon_i \partial \varepsilon_j} d\varepsilon_j + \frac{\partial^2 f}{\partial \varepsilon_i \partial \alpha_j^m} \frac{\partial w}{\partial \chi_j^m} dt$$

### 9.1.3 Development for stress control

Assume $d\sigma_i$ specified.

Obtain compliance matrix

$$C_{ij} = \left( \frac{\partial^2 f}{\partial \varepsilon_i \partial \varepsilon_j} \right)^{-1}$$

Re-arrange increment and substitute compliance matrix

$$\frac{\partial^2 f}{\partial \varepsilon_i \partial \varepsilon_j} d\varepsilon_j = d\sigma_i - \frac{\partial^2 f}{\partial \varepsilon_i \partial \alpha_j^m} d\alpha_j^m$$

$$C_{ki} \frac{\partial^2 f}{\partial \varepsilon_i \partial \varepsilon_j} d\varepsilon_j = d\varepsilon_k = C_{ki} \left( d\sigma_i - \frac{\partial^2 f}{\partial \varepsilon_i \partial \alpha_j^m} d\alpha_j^m \right)$$

Substitute rate of internal variable

$$d\varepsilon_k = C_{ki} \left( d\sigma_i - \frac{\partial^2 f}{\partial \varepsilon_i \partial \alpha_j^m} \frac{\partial w}{\partial \chi_j^m} dt \right)$$

### 9.1.4 Development for general control

Assume control statement of the form:

$$S_{ij} d\sigma_j + E_{ij} d\varepsilon_j = T_i dt$$

Re-arrange control statement and derive modified control (as for rate independent case)

$$S_{ij}\left(\frac{\partial^2 f}{\partial\varepsilon_j\partial\varepsilon_k}d\varepsilon_k+\frac{\partial^2 f}{\partial\varepsilon_j\partial\alpha_k^m}d\alpha_k^m\right)+E_{ik}d\varepsilon_k=T_i dt$$

$$\left(E_{ik}+S_{ij}\frac{\partial^2 f}{\partial\varepsilon_j\partial\varepsilon_k}\right)d\varepsilon_k=T_i dt-S_{ij}\frac{\partial^2 f}{\partial\varepsilon_j\partial\alpha_k^m}d\alpha_k^m$$

$$P_{ik}=\left(E_{ik}+S_{ij}\frac{\partial^2 f}{\partial\varepsilon_j\partial\varepsilon_k}\right)^{-1}$$

$$P_{li}\left(E_{ik}+S_{ij}\frac{\partial^2 f}{\partial\varepsilon_j\partial\varepsilon_k}\right)d\varepsilon_k=d\varepsilon_l=P_{li}\left(T_i dt-S_{ij}\frac{\partial^2 f}{\partial\varepsilon_j\partial\alpha_k^m}d\alpha_k^m\right)$$

Substitute internal variable rate

$$d\varepsilon_l=P_{li}\left(T_i dt-S_{ij}\frac{\partial^2 f}{\partial\varepsilon_j\partial\alpha_k^m}\frac{\partial w}{\partial\chi_k^m}dt\right)$$

## 9.2 Methods based on *g*-*w* functions

### 9.2.1 Basic functions and their derivatives

$$g\left(\sigma_i,\alpha_j^m\right)$$

$$w\left(\chi_i^m,\sigma_j,\alpha_k^n\right)$$

Derivatives

$$\varepsilon_i=-\frac{\partial g}{\partial\sigma_i}$$

$$\chi_i^m=-\frac{\partial g}{\partial\alpha_i^m}$$

Increments

$$d\varepsilon_i=-\frac{\partial^2 g}{\partial\sigma_i\partial\sigma_j}d\sigma_j-\frac{\partial^2 g}{\partial\sigma_i\partial\alpha_j^m}d\alpha_j^m$$

$$d\chi_i^m=-\frac{\partial^2 g}{\partial\alpha_i^m\partial\sigma_j}d\sigma_j-\frac{\partial^2 g}{\partial\alpha_i^m\partial\alpha_j^n}d\alpha_j^n$$

$$d\alpha_i^m=\frac{\partial w}{\partial\chi_i^m}dt$$

### 9.2.2 Development for stress control

Assume $d\sigma_i$ specified.

$$d\alpha_i^m = \frac{\partial w}{\partial \chi_i^m} dt$$

Then

$$d\varepsilon_i = -\frac{\partial^2 g}{\partial \sigma_i \partial \sigma_j} d\sigma_j - \frac{\partial^2 g}{\partial \sigma_i \partial \alpha_j^m} \frac{\partial w}{\partial \chi_j^m} dt$$

### 9.2.3 Development for strain control

Assume $d\varepsilon_i$ specified.

Obtain stiffness matrix

$$D_{ij} = \left( -\frac{\partial^2 g}{\partial \sigma_i \partial \sigma_j} \right)^{-1}$$

Re-arrange increment and substitute stiffness matrix

$$-\frac{\partial^2 g}{\partial \sigma_i \partial \sigma_j} d\sigma_j = d\varepsilon_i + \frac{\partial^2 g}{\partial \sigma_i \partial \alpha_j^m} d\alpha_j^m$$

$$D_{ki} \left( -\frac{\partial^2 g}{\partial \sigma_i \partial \sigma_j} \right) d\sigma_j = d\sigma_k = D_{ki} \left( d\varepsilon_i + \frac{\partial^2 g}{\partial \sigma_i \partial \alpha_j^m} d\alpha_j^m \right)$$

$$d\sigma_k = D_{ki} \left( d\varepsilon_i + \frac{\partial^2 g}{\partial \sigma_i \partial \alpha_j^m} \frac{\partial w}{\partial \chi_j^m} dt \right)$$

### 9.2.4 Development for general control

Assume control statement of the form:

$$S_{ij} d\sigma_j + E_{ij} d\varepsilon_j = T_i dt$$

Re-arrange control statement and derive modified control (as for rate independent case)

$$S_{ik} d\sigma_k + E_{ij} \left( -\frac{\partial^2 g}{\partial \sigma_j \partial \sigma_k} d\sigma_k - \frac{\partial^2 g}{\partial \sigma_j \partial \alpha_k^m} d\alpha_k^m \right) = T_i dt$$

$$\left( S_{ik} - E_{ij} \frac{\partial^2 g}{\partial \sigma_j \partial \sigma_k} \right) d\sigma_k = T_i dt + E_{ij} \frac{\partial^2 g}{\partial \sigma_j \partial \alpha_k^m} d\alpha_k^m$$

$$Q_{ik} = \left( S_{ik} - E_{ij} \frac{\partial^2 g}{\partial \sigma_j \partial \sigma_k} \right)^{-1}$$

$$Q_{li} \left( S_{ik} - E_{ij} \frac{\partial^2 g}{\partial \sigma_j \partial \sigma_k} \right) d\sigma_k = d\sigma_l = Q_{li} \left( T_i dt + E_{ij} \frac{\partial^2 g}{\partial \sigma_j \partial \alpha_k^m} d\alpha_k^m \right)$$

Substitute for internal variable rate

$$d\sigma_l = Q_{li}\left(T_i dt + E_{ij}\frac{\partial^2 g}{\partial\sigma_j \partial\alpha_k^m}\frac{\partial w}{\partial\chi_k^m}dt\right)$$

**Table 2: Current models available THIS TABLE NEEDS REFORMATTING AND UPDATING**

| Model | | | Passes HyperCheck | *f*-form | | | | | *g*-form | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | General control | | | | | General control | | | |
| | | | | Strain control | Stress control | Strain inc. | Stress inc. | Mixed inc. | Strain control | Stress control | Strain inc. | Stress inc. | Mixed inc. | Rate form |
| **Mode 0 models** | | | | | | | | | | | | | | |
| h1e | | | ✓ | ✓ | ✓ | n/a | n/a | n/a | ✓ | ✓ | n/a | n/a | n/a | |
| h1ep | | | ✓ | ✓ | ✗ | n/a | n/a | n/a | ✓ | ✗ | n/a | n/a | n/a | |
| h1epi | | | | | | | | | | | | | | |
| h1epk | | | ✓ | ✓ | ✓ | n/a | n/a | n/a | ✓ | ✓ | n/a | n/a | n/a | ✓ |
| h1epmk_ser | | | ✓ | ✓ | ✓ | n/a | n/a | n/a | ✓ | ✓ | n/a | n/a | n/a | ✓ |
| h1epmk_ser_b | | | ✓ | ✓ | ✓ | n/a | n/a | n/a | ✓ | ✓ | n/a | n/a | n/a | |
| **Mode 1 models** | | | | | | | | | | | | | | |
| h2epmk_ser | | | ✓ | | | ✓ | ✓ | ✓ | | | ✓ | ✓ | ✓ | |
| h2epmk_ser_b | | | ✓ | | | | | ✓ | | | | | ✓ | |

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| hfrict | | | ✓ | ✓ | | | | | ✓ | | | | | |
| hmcc | | | ✓ | ✓ | | | | | ✓ | | | | | |
| | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | |
| | | **Mode 2 models** | | | | | | | | | | | | |
| | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | |