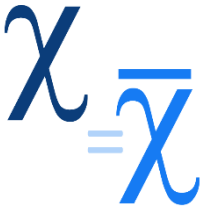


HyperDrive documentation

Version 3.1.0

G.T. Houlsby, May 2024

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, text in *italics* should be substituted by strings or numbers as appropriate. Optional arguments to functions are enclosed in square brackets [ ].

In its basic form HyperDrive requires just two additional supporting files:

- A Python script "*model_file*.py" which provides definitions for the basic functions of hyperplasticity theory for a particular constitutive model (see Section 4).
- A Python script "*test_file*.py" which specifies the test to be simulated (see Section 5).

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 on analytical expressions provided by the user, or determined automatically using the "autograd" package, or determined numerically. (See section 4.3 for further details).
- Strain control, stress control or 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, place HyperDrive.py in an appropriate directory,

Step 2: In the same directory create "*model_file.py*" and "*test_file.py*",

Step 3: Execute "*test_file.py*".

3 Changes from previous versions

- Amalgamation of functions `y_f(chi,eps,alp)` and `y_g(chi,eps,alp)` into a single function `y(eps,sig,alp,chi)`, and of `w_f(chi,eps,alp)` and `w_g(chi,eps,alp)` into a single function `w(eps,sig,alp,chi)`.
- All differentials of the above: `dyde`, `dyds`, `dyda`, `dydc`, `dwdc` have call list `eps,sig,alp,chi`.
- Introduction of weighting function on inner product $\sum_{j,m} \gamma^m \chi_j^m \alpha_j^m$.
- Removal of "modes" 0 and 2 as mode 0 was redundant and mode 2 proved to be problematic.
- Removal of the option to use a "*input_file.dat*" form of test control as this was redundant.
- Removal of "history" option as this is more easily dealt with programatically in Python.
- Addition of large strain options.

4 Specification of "*model_file.py*"

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 = "model_file"`	Omit the .py extension
`name = "Brief description of model"`	
`ndim = …`	Necessary value
`n_int = …`	Necessary value
`n_y = …`	Necessary value
[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 y(eps,sig,alp,chi): …*code*…]	Implements $y = y(\varepsilon,\sigma,\alpha,\chi)$
[def dyde(eps,sig,alp,chi): …*code*…]	Implements $dy/d\varepsilon$
[def dyds(eps,sig,alp,chi): …*code*…]	Implements $dy/d\sigma$
[def dyda(eps,sig,alp,chi): …*code*…]	Implements $dy/d\alpha$
[def dydc(eps,sig,alp,chi): …*code*…]	Implements $dy/d\chi$
[def w(eps,sig,alp,chi): …*code*…]	Implements $w = w(\varepsilon,\sigma,\alpha,\chi)$
[def dwdc(eps,sig,alp,chi): …*code*…]	Implements $dw/d\chi$
[def update(t,eps,sig,alp,chi, dt,deps,dsig,dalp,dchi): …*code*…]	Optional, update certain variables if necessary
[def plot(rec, pname): …*code*…]	Optional special code for plotting

At the start of "*model_file*" the code must set values of (at least) the following variables: ndim, `n_y, n_int.`

Note that functions "*f*" and "*g*" themselves 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 0).

Functions "*w*" and "*dwdc*" are only required if the rate-dependent formulation is used. Conversely, "*y*", "*dyde*", "*dyds*", "*dyda*" and "*dydc*" 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 form "f_form()" is specified (see section 5), then only the functions "deriv", "dfde", "d2fdede", "d2fdeda", "d2fdade", "d2fdada", "y", "dyde", "dyds", "dyda" and "dydc" are required.

Alternatively, if only stress-controlled increments are used or the form "g_form()" is specified (see Section 5), then only the functions "deriv", "dgds", "d2gdsds", "d2gdsda", "d2gdads", "d2gdada", "y", "dyde", "dyds", "dyda" and "dydc" 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()" option, 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 is more accurate than numerical.

4.1 Modes of operation

HyperDrive treats strain and stress variables as vectors and which are implemented as one dimensional arrays `numpy.array(n_dim)`, where `n_dim` is the dimensionality. Internal variables and their corresponding generalised stresses are of the form `numpy.array([n_int,n_dim])`.

General stresses and strains are input and output in "Voigt" notation, but are treated internally using "Mandel" notation.

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

Table 1: dimensionality of principal variables

Variables	Dimension
f , g , w	Scalar
ε , σ , $\frac{\partial f}{\partial \varepsilon}$, $\frac{\partial g}{\partial \sigma}$	array(n_{dim})
$\frac{\partial^2 f}{\partial \varepsilon \partial \varepsilon}$, $\frac{\partial^2 g}{\partial \sigma \partial \sigma}$	array(n_{dim}, n_{dim})
α , χ , $\frac{\partial f}{\partial \alpha}$, $\frac{\partial g}{\partial \alpha}$, $\frac{\partial w}{\partial \chi}$	array(n_{int}, n_{dim})
$\frac{\partial^2 f}{\partial \varepsilon \partial \alpha}$, $\frac{\partial^2 g}{\partial \sigma \partial \alpha}$	array(n_{dim}, n_{int}, n_{dim})
$\frac{\partial^2 f}{\partial \alpha \partial \varepsilon}$, $\frac{\partial^2 g}{\partial \alpha \partial \sigma}$	array(n_{int}, n_{dim}, n_{dim})
$\frac{\partial^2 f}{\partial \alpha \partial \alpha}$, $\frac{\partial^2 g}{\partial \alpha \partial \alpha}$	array(n_{int}, n_{dim}, n_{int}, n_{dim})
y , Λ	array(n_y)
$\frac{\partial y}{\partial \varepsilon}$, $\frac{\partial y}{\partial \sigma}$	array(n_y, n_{dim})
$\frac{\partial y}{\partial \alpha}$, $\frac{\partial y}{\partial \chi}$	array(n_y, n_{int}, n_{dim})

4.2 Example model file

An example model file for a simple model “hnepmk_ser” that illustrates a number of features is given below. This implements a simple multi-surface elastic-plastic model with linear kinematic hardening, for the case of a single or multiple stress and strain variables. Note particularly the dimensionality returned for each of the derivatives.

```
import autograd.numpy as np
from HyperDrive import Utils as hu

check_eps = np.array([0.3,0.05])
check_sig = np.array([8.0,0.5])
check_alp = np.array([[0.2,0.1], [0.18,0.1], [0.16,0.1], [0.14,0.1]])
check_chi = np.array([[0.9,0.1], [1.0,0.1], [0.1,0.1], [1.2,0.1]])

file = "hnepmk_ser"
name = "nD Linear Elastic-Plastic with Multisurface Kinematic Hardening - Series"
const = [2, 100.0, 4, 0.1, 100.0, 0.3, 33.333333, 0.6, 20.0, 1.0, 10.0]

def deriv():
    global ndim, E, k, recip_k, H, name_const
    global n_int, n_inp, n_y, n_const
    ndim  = int(const[0])
    E     = float(const[1])
    n_int = int(const[2])
    n_inp = n_int
```

```
    n_y   = n_int
    n_const = 2 + 2*n_inp
    k = np.array(const[3:3+2*n_inp:2])
    H = np.array(const[4:4+2*n_inp:2])
    recip_k = 1.0 / k
    name_const = ["ndim", "E", "N"]
    for i in range(n_inp):
        name_const.append("k"+str(i+1))
        name_const.append("H"+str(i+1))

def ep(alp): return np.einsum("ni->i",alp)

def f(eps,alp):
    return E*sum((eps-ep(alp))**2)/2.0 + np.einsum("n,ni,ni->",H,alp,alp)/2.0
def dfde(eps,alp):
    return E*(eps - ep(alp))
def dfda(eps,alp):
    return -E*(eps - ep(alp)) + np.einsum("n,ni->ni",H,alp)
def d2fdede(eps,alp):
    return E*np.eye(ndim)
def d2fdeda(eps,alp):
    temp = np.zeros([ndim,n_int,ndim])
    for n in range(n_int):
        temp[:,n,:] = -E*np.eye(ndim)
    return temp
def d2fdade(eps,alp):
    temp = np.zeros([n_int,ndim,ndim])
    for n in range(n_int):
        temp[n,:,:] = -E*np.eye(ndim)
    return temp
def d2fdada(eps,alp):
    temp = np.zeros([n_int,ndim,n_int,ndim])
    for i in range(ndim):
        temp[:,i,:,i] = E
    for n in range(n_inp):
        temp[n,:,n,:] += H[n]*np.eye(ndim)
    return temp

def g(sig,alp):
    return (-sum(sig**2)/(2.0*E) - sum(sig*ep(alp)) +
             np.einsum("n,ni,ni->",H,alp,alp)/2.0)
def dgds(sig,alp):
    return -sig/E - ep(alp)
def dgda(sig,alp):
    return -sig + np.einsum("n,ni->ni",H,alp)
def d2gdsds(sig,alp):
    return -np.eye(ndim) / E
def d2gdsda(sig,alp):
    temp = np.zeros([ndim,n_int,ndim])
    for i in range(ndim):
        temp[i,:,i] = -1.0
    return temp
def d2gdads(sig,alp):
    temp = np.zeros([n_int,ndim,ndim])
    for i in range(ndim):
        temp[:,i,i] = -1.0
    return temp
def d2gdada(sig,alp):
    temp = np.zeros([n_int,ndim,n_int,ndim])
    for n in range(n_int):
        temp[n,:,n,:] = H[n]*np.eye(ndim)
    return temp
```

```
def y(eps,sig,alp,chi):
    return np.sqrt(np.einsum("ni,ni->n",chi,chi))*recip_k - 1.0
def dydc(eps,sig,alp,chi):
    temp = np.zeros([n_y,n_int,ndim])
    for i in range(n_inp):
        t1 = hu.non_zero(np.sqrt(sum(chi[i]**2)))
        temp[i,i,:] = chi[i,:]*recip_k[i] / t1
    return temp
def dyde(eps,sig,alp,chi): return np.zeros([n_y,ndim])
def dyds(eps,sig,alp,chi): return np.zeros([n_y,ndim])
def dyda(eps,sig,alp,chi): return np.zeros([n_y,n_int,ndim])
```

4.3 A note on differentiation methods

Hyperdrive requires just the basic thermodynamic potentials as an absolute minimum – for instance just the "*g*" and "*y*" functions. There are three options available for obtaining the differentials of these functions, which are required to carry out an incremental calculation:

1. Provide explicitly coded forms of the differentials, as shown for instance in the example in section 4.2. Of course, these differentials need to be correctly coded, and sometimes this is not straightforward. That is the main motivation for providing the "check()" routine, which compares the results of the explicitly coded versions with the two other methods available (see below). The explicitly coded versions tend to run fastest, and so would be the preferred option, but they require more programming effort. If all the necessary differentials are provided, then the underlying energy potential (*f* or *g*) is not required in the calculation, and does not need to be provided, except for checking purposes.
2. Numerical differentials are automatically derived by HyperDrive, using finite difference methods. These usually run more slowly than the explicitly coded versions (typically by a factor of about 3), and of course are subject to the usual problems of numerical differentiation – if the numerical difference scheme uses too small a step then round-off errors may become significant, if it is too large then the method becomes less accurate. The main advantage is that explicit coding of the differentials by the user is not required.
3. Automatic differentials are derived by HyperDrive using the "autograd" package. These tend to be slower again than the numerical differentials, by about a further factor of 3. Thus automatic differentials are about an order of magnitude slower than explicitly coded versions. Their advantage over the numerical versions is, however, that there is no loss of accuracy. Apart from the speed of execution, the main disadvantage is that sometimes "autograd" is unable to differentiate a function, depending on how it is coded (and sometimes a simple recoding seems to get around this slightly puzzling problem). If this problem is encountered, error messages may be suppressed (especially when using the "check()" code) by defining (as appropriate) within the model code any of the variables `f_exclude`, `g_exclude`, `y_exclude` or `w_exclude`. No attempt will then be made to use "autograd" on the relevant function. For instance, to exclude y from the process simply include a line such as "`y_exclude = True`" in the main body of the model definition file (**not** within a function definition). The line "`y_exclude = 3`" would, for example, work just as well – it is the fact that the model includes a defined variable `y_exclude` that is detected, not the value of that variable.

5 Commands listed in "*test_file.py*"

Commands should be listed in a Python script "*test_file*.py", where *test_file* is an arbitrary name.

This is a regular Python file, so any line beginning with # is treated as a comment and ignored; blank lines are also ignored. All normal Python functions can be used.

5.1 Specifying the model

title(*title*)

model(*model_file*)

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

const(*constants*)

Provide a Python list [*c1, c2, … etc*] of the constants used in the specified model. Different models require different numbers of constants, see the example model in Section 4.2.

tweak(*const_name, value)*

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

const_from_points(*modtype, points, [Einf=0.0], [epsmax=0.0], [HARM_R=0.0]*)

Derive constants for multisurface model from a set of points.

const_from_curve(*modtype curve npt sigmax HARM_R) NOT FULLY IMPLEMENTED*

const_from_data(*dataname npt maxsig HARM_R) NOT FULLY IMPLEMENTED*

check()

Check the code, see Section 0.

5.2 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, as long as the user-supplied functions are available.

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 fail. For example, if using option "f_form()", the function "dfde" would either have to be supplied by the user, or the function "f" supplied which could be differentiated automatically or numerically.

See further discussion in section 4.3.

voigt()

Use Voigt definitions for stress and strain vectors:

$$\sigma = \left[\sigma_{11} \quad \sigma_{22} \quad \sigma_{33} \quad \tau_{23} \quad \tau_{31} \quad \tau_{12}\right]^T$$

$$\varepsilon = \left[\varepsilon_{11} \quad \varepsilon_{22} \quad \varepsilon_{33} \quad \gamma_{23} \quad \gamma_{31} \quad \gamma_{12}\right]^T$$

See also the discussion in Section 11.

small()

Use small strain definitions (default).

large()

Use large strain definitions. (Working code, but still under development).

hencky()

Use Hencky large strain (default for large strain case).

green()

Use Green large strain.

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 7 for rate-independent incremental algorithms). Default value is 0.5.

quiet()

Suppress some output that indicates the progress of the calculation.

unquiet()

(Default). Unset quiet mode.

5.3 Initialisation and flow control

start()

Start the test.

restart()

Restart a new test without clearing old test from records for plotting *etc*.

init_stress(*stress*)

Specify initial stress in the form of a Python list [sig_1 … sig_{ndim}].

init_strain(*strain*)

Specify initial strain in the form of a Python list [eps_1 … eps_{ndim}].

pause(*[message]*)

Pause output. Hit return to continue. Optional *message* is output.

end()

Stop processing

5.4 Stress and strain increment commands

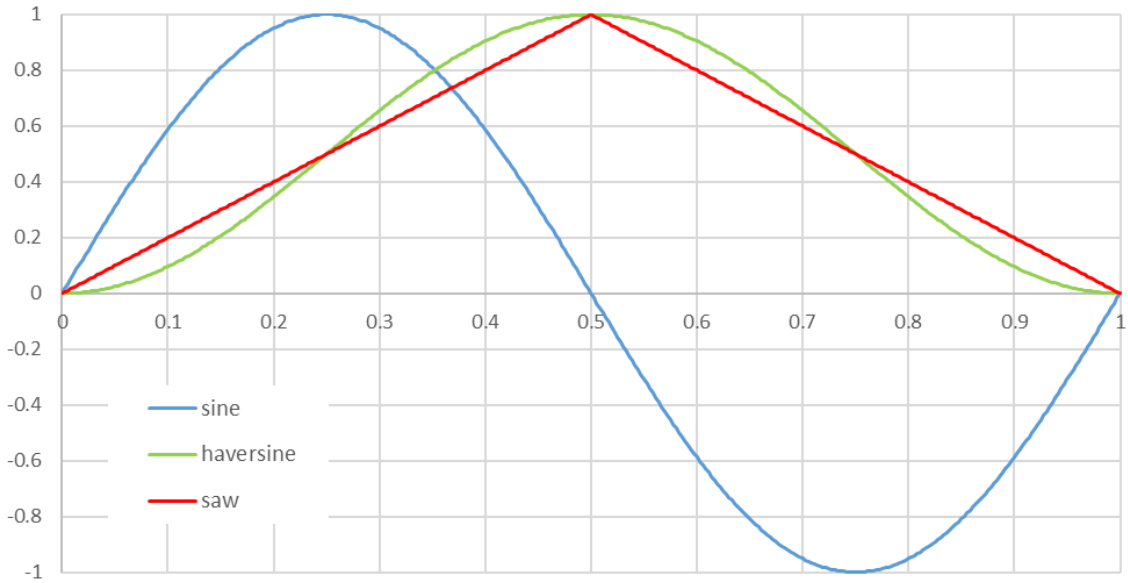
general_inc(*S, E, Tdt, dt, nprint, nsub*)

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, dsig, nprint, nsub)*

Stress controlled increment by *dsig* as a Python list [$dsig_1$, … $dsig_{ndim}$]. Print (or plot) data for *nprint* equal steps, each of which is divided into *nsub* calculation substeps.

strain_inc(*dt, deps, nprint, nsub)*

Strain controlled increment by *deps* as a Python list [$deps_1$, … $deps_{ndim}$]. Print (or plot) data for *nprint* equal steps, each of which is divided into *nsub* calculation substeps.

stress_targ(*dt, sigtarg, nprint, nsub)*

Stress controlled increment to target *sigtarg* as a Python list [$sigtarg_1$, … $sigtarg_{ndim}$]. Print (or plot) data for *nprint* equal steps, each of which is divided into *nsub* calculation substeps.

strain_targ(*dt, epstarg, nprint, nsub)*

Strain controlled increment to target eps*targ* as a Python list [$epstarg_1$, … $epstarg_{ndim}$ Print (or plot) data for *nprint* equal steps, each of which is divided into *nsub* calculation substeps.

stress_cycle(t_{per}, *sigcyc, ctype, ncyc, nprint, nsub)*

Stress controlled cycling *ncyc* times from current stress *sig* to *sig* + *sigcyc* and back to *sig. sigcyc* is specified as a Python list [sigcyc$_1$, sigcyc$_{ndim}$]. 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, ctype, ncyc, nprint, nsub)*

Strain controlled cycling *ncyc* times from current strain *eps* to *eps* + *epscyc* and back to *eps. epscyc* is specified as a Python list [epscyc$_1$, epscyc$_{ndim}$].. 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 *t, eps$_1$ [… eps$_{ndim}$] sig$_1$ […sig$_{ndim}$]*) 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 *t, eps$_1$ […eps$_{ndim}$] sig$_1$ […sig$_{ndim}$]*) 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 *t, eps$_1$ […eps$_{ndim}$]*) 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 *t, sig$_1$ […sig$_{ndim}$]*) and use the stress 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).

v_txl_d(dt, deps1, nprint, nsub)

Run drained triaxial test using Voigt notation.

v_txl_u(dt, deps1, nprint, nsub)

Run drained triaxial test using Voigt notation.

v_dss(dt, dgamma, nprint, nsub)

Run Direct Shear Test using Voigt notation.

5.5 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".

plotCS([*plotfile*])

Plot the results for a *p*,*q* Critical State 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".

graph(*xaxis, yaxis,* [*plotfile,* [*xsize, ysize*]])

Plot a graph, where *xaxis* and *yaxis* are the specified variable names. For example "graph("t", sig_1")" will plot (model with default names) the first 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". If *xsize*, *ysize* are given these specify the plot dimensions in cm (default is 6,6).

specialplot([*plotfile*])

Plot the results using special plotting format for the particular model, if it is provided. 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_file*.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".

csv([*csvfile*])

Print the strains and stresses to file "*csvfile*.csv" (the .csv extension is assumed if not provided). If csvfile not provided output is to "hyper_*model*.csv".

specialprint([*printfile*])

Run the routine "specialprint(printfile, eps, sig, alp, chi)" if it is specified in the model file.

printstate()

Print the current state (eps, sig, alp, chi).

pstress()

Print the current stress.

pstrain()

Print the current strain.

save_state()

Save the current state.

restore_state()

Restore the state to a state saved previously using the “save_state()” command – useful if several tests begin with the same sequence of loading.

5.6 Others

There are some commands still under development that are included in the code: use with great caution only.

5.7 Example input file

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

File entry	**Comment**
`from HyperDrive import Commands as H`	Necessary to access the commands
`H.title('Hyperplasticity test run')`	Title
`H.model('hnepmk_ser')`	Choose model
`H.const([1, 100.0, 4, 0.1, 100.0, 0.3, 33.333333, 0.6, 20.0, 1.0, 10.0])`	Constants override model default values
`H.start()`	Start test
`H.strain_inc(1.0, [0.04], 200, 10)`	Increment strain by 0.4
`H.stress_targ(1.0, [0.0], 100, 10)`	Unload to zero stress
`H.strain_targ(1.0, [0.05], 100, 10)`	Strain to 0.5
`#highlight this section`	A comment – will be ignored
`H.high()`	Set highlighting
`H.stress_inc(1.0, [-1.5], 150, 10)`	Unload by stress increment -1.5
`H.unhigh()`	Unset highlighting
	Blank line will be ignored
`H.stress_cyc(1.0, [1.2], 'saw', 5, 120, 10)`	5 “sawtooth” stress cycles of amplitude 1.2
`H.graph('t', 'sig')`	Graph of stress against time
`H.plot()`	Plot the results
`H.end()`	Finish

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

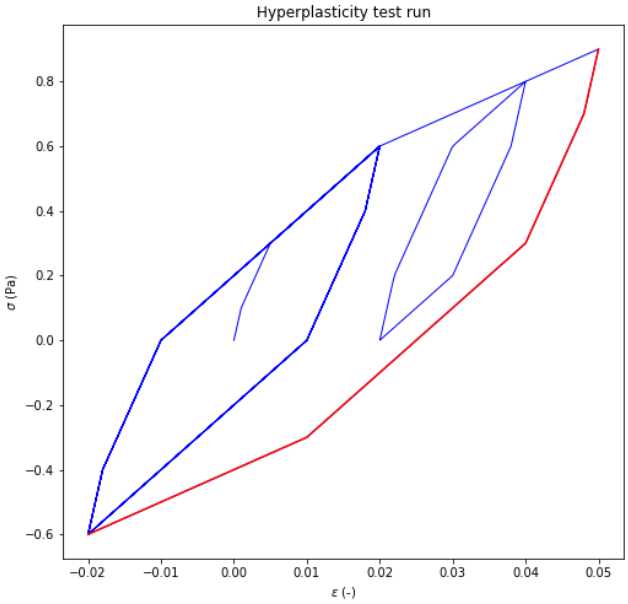


Figure 1: Output figure from example data, note the highlighted section. The cyclic loading at the end of the test is a closed loop.

A slightly more complex version showing how the commands can be interleaved with Python code is shown below.

```
from HyperDrive import Commands as H
import numpy as np
def ran(): return 2.0*(np.random.rand() - 0.5) # random number between -1 and 1
H.title("Test data for NN training")
H.model("hnepmk_ser_b")
H.const([1, 100.0, 4, 0.1, 100.0, 0.3, 33.333333, 0.6, 20.0, 1.0, 10.0])
H.g_form()
H.quiet()
for test in range(10):
    H.start()
    for i in range(10):
        H.stress_targ(1.0, [1.0 * ran()], 50, 10)
        H.stress_inc(1.0, [0.5 * ran()], 50, 10)
    #H.graph("t","eps","t_eps",8,5)
    #H.graph("t","sig","t_sig",8,5)
    H.graph("eps","sig","eps_sig",8,6)
    H.csv('data'+str(test+1))
    #H.pause()
H.end()
```

6 Code checking

A utility routine “check()” is provided, that checks the differentials of the basic functions against the numerically derived and automatically 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 74 checks on model consistency. Note that, if the parameter sets only test cases where yield does not occur, the functions required for plasticity calculations may not be thoroughly checked, as most will just return zeros.

The check routine is invoked by calling “check()” in “*test_file*.py”.

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.

After writing any new user-defined differentials it is advised that the `check()` routine is run, as it is unlikely that faulty code would manage to pass all the tests.

7 Background theory

We base the analyses on the *f-y* or *g-y* formulations of hyperplasticity for the rate-independent case, and the *f-w* or *g-w* forms for the rate independent case. We define strain ε_i, stress σ_i, m internal variables α_j^m and generalized stresses χ_j^m. We define the inner product that defines conjugacy between stress and strain as the conventional $\sum_i \sigma_i \varepsilon_i$ and that for the internal variable and generalised stress as $\sum_{j,m} \gamma^m \chi_j^m \alpha_j^m$, where the weights γ^m are introduced for pragmatic reasons. Any model can be redefined with the values of the weights just set to unity – they are included to simplify the physical interpretation of the internal variable and generalized stress in some cases. In order to preserve the usual summation convention of contraction over a repeated index we shall write $\gamma^m \chi_j^m \equiv \gamma \chi_j^m$, $\gamma^m \alpha_j^m \equiv \gamma \alpha_j^m$, $\frac{1}{\gamma^m}\frac{\partial y^p}{\partial \chi_i^m} \equiv \frac{1}{\gamma}\frac{\partial y^p}{\partial \chi_i^m}$ and $\frac{1}{\gamma^m}\frac{\partial w}{\partial \chi_i^m} \equiv \frac{1}{\gamma}\frac{\partial w}{\partial \chi_i^m}$.

In the following development we highlight the weights in **red**.

8 Notes on incremental forms: rate independent

8.1 Methods based on *f-y* functions

8.1.1 Basic functions and their derivatives

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

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

Derivatives

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

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

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

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

Consistency condition and flow rule during yield

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

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

8.1.2 Development for strain control

Assume $d\varepsilon_i$ specified.

Substitute for generalized stress increment in consistency condition

$$\begin{aligned} -ay_o^p &= \frac{\partial y^p}{\partial \varepsilon_i} d\varepsilon_i + \frac{\partial y^p}{\partial \sigma_i} \left(\frac{\partial^2 f}{\partial \varepsilon_i \partial \varepsilon_j} d\varepsilon_j + \frac{\partial^2 f}{\partial \varepsilon_i \partial \alpha_j^n} d\alpha_j^n \right) + \frac{\partial y^p}{\partial \alpha_i^m} d\alpha_i^m \\ &\quad + \frac{1}{\gamma} \frac{\partial y^p}{\partial \chi_i^m} \left(-\frac{\partial^2 f}{\partial \alpha_i^m \partial \varepsilon_j} d\varepsilon_j - \frac{\partial^2 f}{\partial \alpha_i^m \partial \alpha_j^n} d\alpha_j^n \right) \\ &= \left(\frac{\partial y^p}{\partial \varepsilon_j} + \frac{\partial y^p}{\partial \sigma_i} \frac{\partial^2 f}{\partial \varepsilon_i \partial \varepsilon_j} - \frac{1}{\gamma} \frac{\partial y^p}{\partial \chi_i^m} \frac{\partial^2 f}{\partial \alpha_i^m \partial \varepsilon_j} \right) d\varepsilon_j + \left(\frac{\partial y^p}{\partial \alpha_j^n} + \frac{\partial y^p}{\partial \sigma_i} \frac{\partial^2 f}{\partial \varepsilon_i \partial \alpha_j^n} - \frac{1}{\gamma} \frac{\partial y^p}{\partial \chi_i^m} \frac{\partial^2 f}{\partial \alpha_i^m \partial \alpha_j^n} \right) d\alpha_j^n \end{aligned}$$

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

$$\begin{aligned} &-\left(\frac{\partial y^p}{\partial \alpha_j^n} + \frac{\partial y^p}{\partial \sigma_i} \frac{\partial^2 f}{\partial \varepsilon_i \partial \alpha_j^n} - \frac{1}{\gamma} \frac{\partial y^p}{\partial \chi_i^m} \frac{\partial^2 f}{\partial \alpha_i^m \partial \alpha_j^n} \right) \frac{1}{\gamma} \frac{\partial y^q}{\partial \chi_j^n} \Lambda^q \\ &= ay_o^p + \left(\frac{\partial y^p}{\partial \varepsilon_j} + \frac{\partial y^p}{\partial \sigma_i} \frac{\partial^2 f}{\partial \varepsilon_i \partial \varepsilon_j} - \frac{1}{\gamma} \frac{\partial y^p}{\partial \chi_i^m} \frac{\partial^2 f}{\partial \alpha_i^m \partial \varepsilon_j} \right) d\varepsilon_j \end{aligned}$$

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

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

$$+ \frac{1}{\gamma} \frac{\partial y^p}{\partial \chi_i^m}\left(-\frac{\partial^2 f}{\partial \alpha_i^m \partial \varepsilon_j} d\varepsilon_j - \frac{\partial^2 f}{\partial \alpha_i^m \partial \alpha_j^n} d\alpha_j^n \right)$$

$$= \left(\frac{\partial y^p}{\partial \varepsilon_j} + \frac{\partial y^p}{\partial \sigma_i} \frac{\partial^2 f}{\partial \varepsilon_i \partial \varepsilon_j} - \frac{1}{\gamma} \frac{\partial y^p}{\partial \chi_i^m} \frac{\partial^2 f}{\partial \alpha_i^m \partial \varepsilon_j} \right) C_{jk} \left(d\sigma_k - \frac{\partial^2 f}{\partial \varepsilon_k \partial \alpha_l^n} d\alpha_l^n \right)$$

$$+ \left(\frac{\partial y^p}{\partial \alpha_j^n} + \frac{\partial y^p}{\partial \sigma_i} \frac{\partial^2 f}{\partial \varepsilon_i \partial \alpha_j^n} - \frac{1}{\gamma} \frac{\partial y^p}{\partial \chi_i^m} \frac{\partial^2 f}{\partial \alpha_i^m \partial \alpha_j^n} \right) d\alpha_j^n$$

$$= \left(\frac{\partial y^p}{\partial \varepsilon_j} + \frac{\partial y^p}{\partial \sigma_i} \frac{\partial^2 f}{\partial \varepsilon_i \partial \varepsilon_j} - \frac{1}{\gamma} \frac{\partial y^p}{\partial \chi_i^m} \frac{\partial^2 f}{\partial \alpha_i^m \partial \varepsilon_j} \right) C_{jk} d\sigma_k +$$

$$\left(-\left(\frac{\partial y^p}{\partial \varepsilon_j} + \frac{\partial y^p}{\partial \sigma_i} \frac{\partial^2 f}{\partial \varepsilon_i \partial \varepsilon_j} - \frac{1}{\gamma} \frac{\partial y^p}{\partial \chi_i^m} \frac{\partial^2 f}{\partial \alpha_i^m \partial \varepsilon_j} \right) C_{jk} \frac{\partial^2 f}{\partial \varepsilon_k \partial \alpha_l^n} \right.$$

$$\left. + \left(\frac{\partial y^p}{\partial \alpha_l^n} + \frac{\partial y^p}{\partial \sigma_i} \frac{\partial^2 f}{\partial \varepsilon_i \partial \alpha_l^n} - \frac{1}{\gamma} \frac{\partial y^p}{\partial \chi_i^m} \frac{\partial^2 f}{\partial \alpha_i^m \partial \alpha_l^n} \right) \right) d\alpha_l^n$$

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

$$\left(\left(\frac{\partial y^p}{\partial \varepsilon_j} + \frac{\partial y^p}{\partial \sigma_i} \frac{\partial^2 f}{\partial \varepsilon_i \partial \varepsilon_j} - \frac{1}{\gamma} \frac{\partial y^p}{\partial \chi_i^m} \frac{\partial^2 f}{\partial \alpha_i^m \partial \varepsilon_j} \right) C_{jk} \frac{\partial^2 f}{\partial \varepsilon_k \partial \alpha_l^n} \right.$$

$$\left. - \left(\frac{\partial y^p}{\partial \alpha_l^n} + \frac{\partial y^p}{\partial \sigma_i} \frac{\partial^2 f}{\partial \varepsilon_i \partial \alpha_l^n} - \frac{1}{\gamma} \frac{\partial y^p}{\partial \chi_i^m} \frac{\partial^2 f}{\partial \alpha_i^m \partial \alpha_l^n} \right) \right) \frac{1}{\gamma} \frac{\partial y^q}{\partial \chi_l^n} \Lambda^q$$

$$= ay_o^p + \left(\frac{\partial y^p}{\partial \varepsilon_j} + \frac{\partial y^p}{\partial \sigma_i} \frac{\partial^2 f}{\partial \varepsilon_i \partial \varepsilon_j} - \frac{1}{\gamma} \frac{\partial y^p}{\partial \chi_i^m} \frac{\partial^2 f}{\partial \alpha_i^m \partial \varepsilon_j} \right) C_{jk} d\sigma_k$$

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}
-a y_o^p &= \frac{\partial y^p}{\partial \varepsilon_i} d\varepsilon_i + \frac{\partial y^p}{\partial \sigma_i} \left(\frac{\partial^2 f}{\partial \varepsilon_i \partial \varepsilon_j} d\varepsilon_j + \frac{\partial^2 f}{\partial \varepsilon_i \partial \alpha_j^n} d\alpha_j^n \right) + \frac{\partial y^p}{\partial \alpha_i^m} d\alpha_i^m \\
&\quad + \frac{1}{\gamma} \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) \\
&= \left(\frac{\partial y^p}{\partial \varepsilon_j} + \frac{\partial y^p}{\partial \sigma_i} \frac{\partial^2 f}{\partial \varepsilon_i \partial \varepsilon_j} - \frac{1}{\gamma} \frac{\partial y^p}{\partial \chi_i^m} \frac{\partial^2 f}{\partial \alpha_i^m \partial \varepsilon_j} \right) d\varepsilon_j \\
&\quad + \left(\frac{\partial y^p}{\partial \alpha_j^n} + \frac{\partial y^p}{\partial \sigma_i} \frac{\partial^2 f}{\partial \varepsilon_i \partial \alpha_j^n} - \frac{1}{\gamma} \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 \sigma_i} \frac{\partial^2 f}{\partial \varepsilon_i \partial \varepsilon_j} - \frac{1}{\gamma} \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) \\
&\quad + \left(\frac{\partial y^p}{\partial \alpha_j^n} + \frac{\partial y^p}{\partial \sigma_i} \frac{\partial^2 f}{\partial \varepsilon_i \partial \alpha_j^n} - \frac{1}{\gamma} \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 \sigma_i} \frac{\partial^2 f}{\partial \varepsilon_i \partial \varepsilon_j} - \frac{1}{\gamma} \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 \sigma_i} \frac{\partial^2 f}{\partial \varepsilon_i \partial \varepsilon_j} - \frac{1}{\gamma} \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} \right. \\
&\quad \left. + \left(\frac{\partial y^p}{\partial \alpha_k^n} + \frac{\partial y^p}{\partial \sigma_i} \frac{\partial^2 f}{\partial \varepsilon_i \partial \alpha_k^n} - \frac{1}{\gamma} \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

$$\left(\left(\frac{\partial y^p}{\partial \varepsilon_j}+\frac{\partial y^p}{\partial \sigma_i}\frac{\partial^2 f}{\partial \varepsilon_i \partial \varepsilon_j}-\frac{1}{\gamma}\frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 f}{\partial \alpha_i^m \partial \varepsilon_j}\right)P_{jr}S_{rl}\frac{\partial^2 f}{\partial \varepsilon_l \partial \alpha_k^n}\right.$$
$$\left.-\left(\frac{\partial y^p}{\partial \alpha_k^n}+\frac{\partial y^p}{\partial \sigma_i}\frac{\partial^2 f}{\partial \varepsilon_i \partial \alpha_k^n}-\frac{1}{\gamma}\frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 f}{\partial \alpha_i^m \partial \alpha_k^n}\right)\right)\frac{1}{\gamma}\frac{\partial y^q}{\partial \chi_k^n}\Lambda^q =$$
$$ay_o^p+\left(\frac{\partial y^p}{\partial \varepsilon_j}+\frac{\partial y^p}{\partial \sigma_i}\frac{\partial^2 f}{\partial \varepsilon_i \partial \varepsilon_j}-\frac{1}{\gamma}\frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 f}{\partial \alpha_i^m \partial \varepsilon_j}\right)P_{jr}T_r dt$$

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

8.2.1 Basic functions and their derivatives

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

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

Derivatives

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

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

Increments

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

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

Consistency condition and flow rule during yield

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

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

8.2.2 Development for stress control

Assume $d\sigma_i$ specified.

Substitute for generalized stress increment in consistency condition

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

$$+\frac{1}{\gamma}\frac{\partial y^p}{\partial \chi_i^m}\left(-\frac{\partial^2 g}{\partial \alpha_i^m \partial \sigma_j} d\sigma_j - \frac{\partial^2 g}{\partial \alpha_i^m \partial \alpha_j^n} d\alpha_j^n\right)$$

$$= \left(\frac{\partial y^p}{\partial \sigma_j} - \frac{\partial y^p}{\partial \varepsilon_i}\frac{\partial^2 g}{\partial \sigma_i \partial \sigma_j} - \frac{1}{\gamma}\frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 g}{\partial \alpha_i^m \partial \sigma_j}\right) d\sigma_j + \left(\frac{\partial y^p}{\partial \alpha_j^n} - \frac{\partial y^p}{\partial \varepsilon_i}\frac{\partial^2 g}{\partial \sigma_i \partial \alpha_j^m} - \frac{1}{\gamma}\frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 g}{\partial \alpha_i^m \partial \alpha_j^n}\right) d\alpha_j^n$$

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

$$-\left(\frac{\partial y^p}{\partial \alpha_j^n} - \frac{\partial y^p}{\partial \varepsilon_i}\frac{\partial^2 g}{\partial \sigma_i \partial \alpha_j^m} - \frac{1}{\gamma}\frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 g}{\partial \alpha_i^m \partial \alpha_j^n}\right)\frac{1}{\gamma}\frac{\partial y^q}{\partial \chi_j^n}\Lambda^q$$

$$= ay_o^p + \left(\frac{\partial y^p}{\partial \sigma_j} - \frac{\partial y^p}{\partial \varepsilon_i}\frac{\partial^2 g}{\partial \sigma_i \partial \sigma_j} - \frac{1}{\gamma}\frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 g}{\partial \alpha_i^m \partial \sigma_j}\right) d\sigma_j$$

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

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

$$+\frac{1}{\gamma}\frac{\partial y^p}{\partial \chi_i^m}\left(-\frac{\partial^2 g}{\partial \alpha_i^m \partial \sigma_j}d\sigma_j - \frac{\partial^2 g}{\partial \alpha_i^m \partial \alpha_j^n}d\alpha_j^n\right)$$

$$=\left(\frac{\partial y^p}{\partial \sigma_j} - \frac{\partial y^p}{\partial \varepsilon_i}\frac{\partial^2 g}{\partial \sigma_i \partial \sigma_j} - \frac{1}{\gamma}\frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 g}{\partial \alpha_i^m \partial \sigma_j}\right)D_{jk}\left(d\varepsilon_k + \frac{\partial^2 g}{\partial \sigma_k \partial \alpha_l^n}d\alpha_l^n\right)$$

$$+\left(\frac{\partial y^p}{\partial \alpha_j^n} - \frac{\partial y^p}{\partial \varepsilon_i}\frac{\partial^2 g}{\partial \sigma_i \partial \alpha_j^n} - \frac{1}{\gamma}\frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 g}{\partial \alpha_i^m \partial \alpha_j^n}\right)d\alpha_j^n$$

$$=\left(\frac{\partial y^p}{\partial \sigma_j} - \frac{\partial y^p}{\partial \varepsilon_i}\frac{\partial^2 g}{\partial \sigma_i \partial \sigma_j} - \frac{1}{\gamma}\frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 g}{\partial \alpha_i^m \partial \sigma_j}\right)D_{jk}d\varepsilon_k +$$

$$\left(\left(\frac{\partial y^p}{\partial \sigma_j} - \frac{\partial y^p}{\partial \varepsilon_i}\frac{\partial^2 g}{\partial \sigma_i \partial \sigma_j} - \frac{1}{\gamma}\frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 g}{\partial \alpha_i^m \partial \sigma_j}\right)D_{jk}\frac{\partial^2 g}{\partial \sigma_k \partial \alpha_l^n} + \left(\frac{\partial y^p}{\partial \alpha_l^n} - \frac{\partial y^p}{\partial \varepsilon_i}\frac{\partial^2 g}{\partial \sigma_i \partial \alpha_l^n} - \frac{1}{\gamma}\frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 g}{\partial \alpha_i^m \partial \alpha_l^n}\right)\right)d\alpha_l^n$$

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

$$-\left(\left(\frac{\partial y^p}{\partial \sigma_j} - \frac{\partial y^p}{\partial \varepsilon_i}\frac{\partial^2 g}{\partial \sigma_i \partial \sigma_j} - \frac{1}{\gamma}\frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 g}{\partial \alpha_i^m \partial \sigma_j}\right)D_{jk}\frac{\partial^2 g}{\partial \sigma_k \partial \alpha_l^n} + \left(\frac{\partial y^p}{\partial \alpha_l^n} - \frac{\partial y^p}{\partial \varepsilon_i}\frac{\partial^2 g}{\partial \sigma_i \partial \alpha_l^n} - \frac{1}{\gamma}\frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 g}{\partial \alpha_i^m \partial \alpha_l^n}\right)\right)\frac{1}{\gamma}\frac{\partial y^q}{\partial \chi_l^n}\Lambda^q$$

$$= ay_o^p + \left(\frac{\partial y^p}{\partial \sigma_j} - \frac{\partial y^p}{\partial \varepsilon_i}\frac{\partial^2 g}{\partial \sigma_i \partial \sigma_j} - \frac{1}{\gamma}\frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 g}{\partial \alpha_i^m \partial \sigma_j}\right)D_{jk}d\varepsilon_k$$

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

$$-a y_o^p = \frac{1}{\gamma} \frac{\partial y^p}{\partial \chi_i^m} \left(-\frac{\partial^2 g}{\partial \alpha_i^m \partial \sigma_j} d\sigma_j - \frac{\partial^2 g}{\partial \alpha_i^m \partial \alpha_j^n} d\alpha_j^n \right)$$

$$+ \frac{\partial y^p}{\partial \varepsilon_l} \left(-\frac{\partial^2 g}{\partial \sigma_l \partial \sigma_j} d\sigma_j - \frac{\partial^2 g}{\partial \sigma_l \partial \alpha_j^n} d\alpha_j^n \right) + \frac{\partial y^p}{\partial \sigma_j} d\sigma_j + \frac{\partial y^p}{\partial \alpha_i^m} d\alpha_i^m$$

$$= \left(\frac{\partial y^p}{\partial \sigma_j} - \frac{\partial y^p}{\partial \varepsilon_l} \frac{\partial^2 g}{\partial \sigma_l \partial \sigma_j} - \frac{1}{\gamma} \frac{\partial y^p}{\partial \chi_i^m} \frac{\partial^2 g}{\partial \alpha_i^m \partial \sigma_j} \right) d\sigma_j$$

$$+ \left(\frac{\partial y^p}{\partial \alpha_j^n} - \frac{\partial y^p}{\partial \varepsilon_l} \frac{\partial^2 g}{\partial \sigma_l \partial \alpha_j^n} - \frac{1}{\gamma} \frac{\partial y^p}{\partial \chi_i^m} \frac{\partial^2 g}{\partial \alpha_i^m \partial \alpha_j^n} \right) d\alpha_j^n$$

$$= \left(\frac{\partial y^p}{\partial \sigma_j} - \frac{\partial y^p}{\partial \varepsilon_l} \frac{\partial^2 g}{\partial \sigma_l \partial \sigma_j} - \frac{1}{\gamma} \frac{\partial y^p}{\partial \chi_i^m} \frac{\partial^2 g}{\partial \alpha_i^m \partial \sigma_j} \right) Q_{jr} \left(T_r dt + E_{rl} \frac{\partial^2 g}{\partial \sigma_l \partial \alpha_k^m} d\alpha_k^m \right)$$

$$+ \left(\frac{\partial y^p}{\partial \alpha_j^n} - \frac{\partial y^p}{\partial \varepsilon_l} \frac{\partial^2 g}{\partial \sigma_l \partial \alpha_j^n} - \frac{1}{\gamma} \frac{\partial y^p}{\partial \chi_i^m} \frac{\partial^2 g}{\partial \alpha_i^m \partial \alpha_j^n} \right) d\alpha_j^n$$

$$= \left(\frac{\partial y^p}{\partial \sigma_j} - \frac{\partial y^p}{\partial \varepsilon_l} \frac{\partial^2 g}{\partial \sigma_l \partial \sigma_j} - \frac{1}{\gamma} \frac{\partial y^p}{\partial \chi_i^m} \frac{\partial^2 g}{\partial \alpha_i^m \partial \sigma_j} \right) Q_{jr} T_r dt$$

$$+ \left(\left(\frac{\partial y^p}{\partial \sigma_j} - \frac{\partial y^p}{\partial \varepsilon_l} \frac{\partial^2 g}{\partial \sigma_l \partial \sigma_j} - \frac{1}{\gamma} \frac{\partial y^p}{\partial \chi_i^m} \frac{\partial^2 g}{\partial \alpha_i^m \partial \sigma_j} \right) Q_{jr} E_{rl} \frac{\partial^2 g}{\partial \sigma_l \partial \alpha_k^n} \right) d\alpha_k^n$$

$$+ \left(\frac{\partial y^p}{\partial \alpha_k^n} - \frac{\partial y^p}{\partial \varepsilon_l} \frac{\partial^2 g}{\partial \sigma_l \partial \alpha_k^n} - \frac{1}{\gamma} \frac{\partial y^p}{\partial \chi_i^m} \frac{\partial^2 g}{\partial \alpha_i^m \partial \alpha_k^n} \right)$$

Re-arrange to solve for plastic multipliers

$$\left(\begin{array}{l} -\left(\frac{\partial y^p}{\partial \sigma_j} - \frac{\partial y^p}{\partial \varepsilon_l}\frac{\partial^2 g}{\partial \sigma_l \partial \sigma_j} - \frac{1}{\gamma}\frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 g}{\partial \alpha_i^m \partial \sigma_j}\right) Q_{jr} E_{rl} \frac{\partial^2 g}{\partial \sigma_l \partial \alpha_k^n} \\ -\left(\frac{\partial y^p}{\partial \alpha_k^n} - \frac{\partial y^p}{\partial \varepsilon_l}\frac{\partial^2 g}{\partial \sigma_l \partial \alpha_k^n} - \frac{1}{\gamma}\frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 g}{\partial \alpha_i^m \partial \alpha_k^n}\right) \end{array}\right) \frac{1}{\gamma}\frac{\partial y^q}{\partial \chi_k^n}\Lambda^q$$

$$= a y_o^p + \left(\frac{\partial y^p}{\partial \sigma_j} - \frac{\partial y^p}{\partial \varepsilon_l}\frac{\partial^2 g}{\partial \sigma_l \partial \sigma_j} - \frac{1}{\gamma}\frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 g}{\partial \alpha_i^m \partial \sigma_j}\right) Q_{jr} T_r dt$$

8.2.5 Stiffness matrix calculation

Although this is not required for HyperDrive, this is a useful result for other programming applications. We start from the following results, drawn from the above:

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

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

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

If $y^p < 0$, $\Lambda^p = 0$, otherwise:

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

And after substitutions:

$$-\left(\frac{\partial y^p}{\partial \alpha_j^n} - \frac{\partial y^p}{\partial \varepsilon_i}\frac{\partial^2 g}{\partial \sigma_i \partial \alpha_j^n} - \frac{1}{\gamma}\frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 g}{\partial \alpha_i^m \partial \alpha_j^n}\right)\frac{1}{\gamma}\frac{\partial y^q}{\partial \chi_j^n}\Lambda^q$$

$$= a y_o^p + \left(\frac{\partial y^p}{\partial \sigma_j} - \frac{\partial y^p}{\partial \varepsilon_i}\frac{\partial^2 g}{\partial \sigma_i \partial \sigma_j} - \frac{1}{\gamma}\frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 g}{\partial \alpha_i^m \partial \sigma_j}\right) d\sigma_j$$

We re-write this for $y_o^p = 0$:

$$-\left(\frac{\partial y^p}{\partial \alpha_j^n} - \frac{\partial y^p}{\partial \varepsilon_i}\frac{\partial^2 g}{\partial \sigma_i \partial \alpha_j^n} - \frac{1}{\gamma}\frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 g}{\partial \alpha_i^m \partial \alpha_j^n}\right)\frac{1}{\gamma}\frac{\partial y^q}{\partial \chi_j^n}\Lambda^q$$

$$= \left(\frac{\partial y^p}{\partial \sigma_j} - \frac{\partial y^p}{\partial \varepsilon_i}\frac{\partial^2 g}{\partial \sigma_i \partial \sigma_j} - \frac{1}{\gamma}\frac{\partial y^p}{\partial \chi_i^m}\frac{\partial^2 g}{\partial \alpha_i^m \partial \sigma_j}\right) d\sigma_j$$

Which for convenience we write:

$$A^{pq}\Lambda^q = B_j^p d\sigma_j$$

Where $A^{pq} = -\left(\frac{\partial y^p}{\partial \alpha_j^n} - \frac{\partial y^p}{\partial \varepsilon_i} \frac{\partial^2 g}{\partial \sigma_i \partial \alpha_j^n} - \frac{1}{\gamma} \frac{\partial y^p}{\partial \chi_i^m} \frac{\partial^2 g}{\partial \alpha_i^m \partial \alpha_j^n} \right) \frac{1}{\gamma} \frac{\partial y^q}{\partial \chi_j^n}$

and $B_j^p = \left(\frac{\partial y^p}{\partial \sigma_j} - \frac{\partial y^p}{\partial \varepsilon_i} \frac{\partial^2 g}{\partial \sigma_i \partial \sigma_j} - \frac{1}{\gamma} \frac{\partial y^p}{\partial \chi_i^m} \frac{\partial^2 g}{\partial \alpha_i^m \partial \sigma_j} \right)$

However, if $y^p < 0$, $\Lambda^p = 0$, and in this case we replace the relevant lines in the above matrices by:

$$A^{pq} = 1, p = q \quad A^{pq} = 0, p \neq q$$

$$B_j^p = 0, \forall j$$

Using these modified forms we obtain $\Lambda^q = \left(A^{pq} \right)^{-1} B_j^p d\sigma_j$

and then obtain

$$d\varepsilon_i = \left[-\frac{\partial^2 g}{\partial \sigma_i \partial \sigma_j} - \frac{\partial^2 g}{\partial \sigma_i \partial \alpha_k^m} \frac{1}{\gamma} \frac{\partial y^q}{\partial \chi_k^m} \left(A^{pq} \right)^{-1} B_j^p \right] d\sigma_j$$

We invert the bracketed term to obtain the elastic-plastic stiffness matrix.

9 Notes on incremental forms: rate dependent

9.1 Methods based on *f-w* functions

9.1.1 Basic functions and their derivatives

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

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

Derivatives

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

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

Increments

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

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

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

9.1.2 Development for strain control

Assume $d\varepsilon_i$ specified.

$$d\alpha_i^m = \frac{1}{\gamma}\frac{\partial w}{\partial \chi_i^m}dt$$

Then

$$d\sigma_i = \frac{\partial^2 f}{\partial \varepsilon_i \partial \varepsilon_j}d\varepsilon_j + \frac{\partial^2 f}{\partial \varepsilon_i \partial \alpha_j^m}\frac{1}{\gamma}\frac{\partial w}{\partial \chi_j^m}dt$$

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

$$d\varepsilon_k = C_{ki}\left(d\sigma_i - \frac{\partial^2 f}{\partial \varepsilon_i \partial \alpha_j^m}\frac{1}{\gamma}\frac{\partial w}{\partial \chi_j^m}dt\right)$$

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

$$d\varepsilon_l=P_{li}\left(T_i dt-S_{ij}\frac{\partial^2 f}{\partial\varepsilon_j\partial\alpha_k^m}\frac{1}{\gamma}\frac{\partial w}{\partial\chi_k^m}dt\right)$$

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

9.2.1 Basic functions and their derivatives

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

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

Derivatives

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

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

Increments

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

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

$$d\alpha_i^m=\frac{1}{\gamma}\frac{\partial w}{\partial\chi_i^m}dt$$

9.2.2 Development for stress control

Assume $d\sigma_i$ specified.

$$d\alpha_i^m=\frac{1}{\gamma}\frac{\partial w}{\partial\chi_i^m}dt$$

Then

$$d\varepsilon_i=-\frac{\partial^2 g}{\partial\sigma_i\partial\sigma_j}d\sigma_j-\frac{\partial^2 g}{\partial\sigma_i\partial\alpha_j^m}\frac{1}{\gamma}\frac{\partial w}{\partial\chi_j^m}dt$$

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

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

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

$$d\sigma_k = D_{ki}\left(d\varepsilon_i + \frac{\partial^2 g}{\partial \sigma_i \partial \alpha_j^m} \frac{1}{\gamma} \frac{\partial w}{\partial \chi_j^m} dt \right)$$

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

$$d\sigma_l = Q_{li}\left(T_i dt + E_{ij} \frac{\partial^2 g}{\partial \sigma_j \partial \alpha_k^m} \frac{1}{\gamma} \frac{\partial w}{\partial \chi_k^m} dt \right)$$

10 Utility functions provided in HyperDrive

The following utility functions are provided to allow straightforward definition of the different potential functions. They are accessed by including the line "`from HyperDrive import Utils as hu`" in the model definition file, and then can be used as "`hu.function()`".

This section is incomplete work in progress.

```
mac(x): # Macaulay bracket
macm(x, delta = 0.0001): # Macaulay bracket, with possible rounding
```

```
S(x): # modified Signum function
non_zero(x): # return a finite small value if argument close to zero
Ineg(x): # Indicator function for set of negative reals
Nneg(x): # Normal Cone for set of negative reals
w_rate_lin(y, mu):
w_rate_lind(y, mu):
w_rate_rpt(y, mu,r):
w_rate_rptd(y, mu,r):
# Tensor utilities
    delta # Kronecker delta, unit tensor
dev(t): # deviator
trans(t): # transpose
sym(t): # symmetric part
skew(t): # skew (antisymmetric) part
tr1_o(t): # trace
tr1(t): # trace
tr2_o(t): # trace of square
tr2(t): # trace of square
tr3_o(t): # trace of cube
tr3(t): # trace of cube
i1(t): # 1st invariant
i2(t): # 2nd invariant (NB some sources use this with opposite sign)
i3(t): # 3rd invariant
j2_o(t): # 2nd invariant of deviator
j2(t): # 2nd invariant of deviator
j3_o(t): # 2nd invariant of deviator
j3(t): # 3rd invariant of deviator
det(t): # determinant
iprod(t1,t2): # inner product of two tensors
cont(t1,t2): # double contraction of two tensors
    # shorthand for 4th order products
pijkl(t1,t2): return np.einsum("ij,kl->ijkl",t1,t2)
pikjl(t1,t2): return np.einsum("ik,jl->ijkl",t1,t2)
piljk(t1,t2): return np.einsum("il,jk->ijkl",t1,t2)
    II     = pikjl(delta, delta)  # 4th order unit tensor, contracts with t to
give t
    IIb    = piljk(delta, delta)  # 4th order unit tensor, contracts with t to
give t-transpose
    IIbb   = pijkl(delta, delta)  # 4th order unit tensor, contracts with t to
give tr(t).delta
    IIsym = (II + IIb) / 2.0     # 4th order unit tensor (symmetric)
    PP    = II    - (IIbb / 3.0) # 4th order projection tensor
    PPb   = IIb   - (IIbb / 3.0) # 4th order projection tensor
    PPsym = IIsym - (IIbb / 3.0) # 4th order projection tensor (symmetric)
    # mixed invariants
trm_ab(a,b) :
trm_a2b(a,b):
trm_ab2(a,b):
trm_a2b2(a,b):
    # Mandel notation utilities
    delta_m = np.array([1.0, 1.0, 1.0, 0.0, 0.0, 0.0])
dev_m(t): # deviator
cont_m(t1,t2): # contraction
    #conversions
t_to_ve(t): return np.array([t[0,0], t[1,1], t[2,2],
t_to_vs(t): return np.array([t[0,0], t[1,1], t[2,2],
```

```
ve_to_t(t): return np.array([[    t[0], 0.5*t[5], 0.5*t[4]],
vs_to_t(t): return np.array([[t[0], t[5], t[4]],
    # traces and invariants
tr1_m(t):  return t[0] + t[1] + t[2] # Voigt trace
tr2_m(t): return t[0]**2 + t[1]**2 + t[2]**2 + \
tr3_m(t):
i1_v(t): # 1st invariant
i2_m(t): # 2nd invariant
i3_m(t): # 3rd invariant
j2_m(t): # 2nd invariant of deviator
j3_m(t): # 3rd invariant of deviator
    # derivatives of invariants
    di1_m  = delta_v
di2_m(t):
di3_m(t):
dj2_m(t):
pij_m(t1,t2): return np.einsum("i,j->ij",t1,t2)
    d2j2_m = np.eye(6) - II_v/3.0
det_m(t): # determinant of strain-like
    #mixed invariants and their derivatives (all assumed stress-like)
trm_ab_m(a,b):
trm_ab2_m(a,b):
dtrm_ab_a_m(a):
dtrm_ab2_a_m(b):
dtrm_ab2b_a_m(a,b):
```

11 Voigt and Mandel notation

Stresses and strains are best described for theoretical analysis a second order tensors, written for the purposes of this discussion in subscript notation:

$$\boldsymbol{\sigma} = \sigma_{ij} = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{23} \end{bmatrix}$$

$$\boldsymbol{\varepsilon} = \varepsilon_{ij} = \begin{bmatrix} \varepsilon_{11} & \varepsilon_{12} & \varepsilon_{13} \\ \varepsilon_{21} & \varepsilon_{22} & \varepsilon_{23} \\ \varepsilon_{31} & \varepsilon_{32} & \varepsilon_{23} \end{bmatrix}$$

Both tensors are symmetric.

For computational purposes it is inefficient to work in terms of the 9 tensor variables, but instead exploit the symmetry of the tensors and only work in terms of 6 variables. This is achieved conventionally by Voigt notation, in which the tensors are represented by vectors of length 6:

$$\sigma_i^v = \begin{bmatrix} \sigma_{11} & \sigma_{22} & \sigma_{33} & \tau_{23} & \tau_{31} & \tau_{12} \end{bmatrix}^T = \boldsymbol{\sigma}^{vT}$$

$$\varepsilon_i^v = \begin{bmatrix} \varepsilon_{11} & \varepsilon_{22} & \varepsilon_{33} & \gamma_{23} & \gamma_{31} & \gamma_{12} \end{bmatrix}^T = \boldsymbol{\varepsilon}^{vT}$$

Where shear stresses and engineering shear strains are used:

$$\tau_{23} = \sigma_{23}, \tau_{31} = \sigma_{31}, \tau_{12} = \sigma_{12}$$

$$\gamma_{23} = 2\varepsilon_{23}, \gamma_{31} = 2\varepsilon_{31}, \gamma_{12} = 2\varepsilon_{12}$$

There appears to be no standardised ordering for the three shear terms, and common FE programs for instance use different orderings from the above.

Importantly, these definitions allow simple definition of the contraction:

$$\boldsymbol{\sigma} : \boldsymbol{\varepsilon} = \sigma_{ij}\varepsilon_{ij} = \sigma_i^v \varepsilon_i^v = \boldsymbol{\sigma}^{v\mathrm{T}} \boldsymbol{\varepsilon}^v$$

However, in the development of many models (for instance in hyperplasticity), much use is made of invariants of the stress and strain tensors, and first and second derivatives of those invariants. A problem arises with the Voigt notation that, because of the factor of 2 involved in the definition of the engineering shear strains, the traces and invariants (except $\mathrm{tr}(\mathbf{t}) = t_{ii}$) involve different formula depending on whether the tensor is stress-like or strain-like.

Not only does this involve additional programming effort, but it also requires meticulous attention to detail as to whether invariants (including mixed invariants) are functions of stress-like or strain-like tensors.

This problem is circumvented by adopting an alternative notation, which appears to be by Mandel, but has not been commonly adopted in the FE literature. The Mandel notation uses:

$$\sigma_i^m = \begin{bmatrix} \sigma_{11} & \sigma_{22} & \sigma_{33} & \sqrt{2}\tau_{23} & \sqrt{2}\tau_{31} & \sqrt{2}\tau_{12} \end{bmatrix}^{\mathrm{T}} = \boldsymbol{\sigma}^{m\mathrm{T}}$$

$$\varepsilon_i^m = \begin{bmatrix} \varepsilon_{11} & \varepsilon_{22} & \varepsilon_{33} & \sqrt{2}\varepsilon_{23} & \sqrt{2}\varepsilon_{31} & \sqrt{2}\varepsilon_{12} \end{bmatrix}^{\mathrm{T}} = \boldsymbol{\varepsilon}^{m\mathrm{T}}$$

Importantly, these definitions again allow simple definition of the contraction:

$$\boldsymbol{\sigma} : \boldsymbol{\varepsilon} = \sigma_{ij}\varepsilon_{ij} = \sigma_i^m \varepsilon_i^m = \boldsymbol{\sigma}^{m\mathrm{T}} \boldsymbol{\varepsilon}^m$$

In the Mandel notation the invariants take the same form for any tensor.

In HyperDrive conventional Voigt notation is used for input and output of general stresses and strains, but these are converted to Mandel notation internally for computation. Importantly, this means that any manipulation of stresses and strains within *model_file*.py should take account of the fact that Mandel notation is being used.