BP yield function - public code to be published with the paper

Supplementary material to the paper:

S. Stupkiewicz, R. Denzer, A. Piccolroaz and D. Bigoni
Implicit yield function formulation for granular and rock-like materials
Computational Mechanics (2014)
doi:10.1007/s00466-014-1047-8

Corresponding author:

Stanislaw Stupkiewicz
Institute of Fundamental Technological Research (IPPT PAN), Warsaw, Poland
sstupkie@ippt.pan.pl
http://www.ippt.pan.pl/~sstupkie

The file can be downloaded from:

http://www.ippt.pan.pl/~sstupkie/files/BPyield.html

version 1.2 (2014-06-11)

Description

This *Mathematica* notebook contains the *AceGen* [1] input for generating the numerical code (subroutine) that computes the implicit BP yield function and its first and second derivatives. The complete implicit yield function formulation is provided in [2]. The original BP yield function, which, in its original form, is not easily applicable in FE computations, is described in detail in [3].

[1] J. Korelc, *AceGen* code generation system, http://www.fgg.uni-lj.si/Symech/.
[2] S. Stupkiewicz, R.P. Denzer, A. Piccolroaz and D. Bigoni, Implicit yield function formulation for granular and rock-like materials, *Computational Mechanics* (2014), doi:10.1007/s00466-014-1047-8.
[3] D. Bigoni and A. Piccolroaz, Yield criteria for quasibrittle and frictional materials, *Int. J. Sol. Struct.* 41, 2855-2878, 2004.

The input parameters are the following:

sigma$$ - components of stress tensor in the Kelvin notation, i.e. $\{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5, \sigma_6\} = \left\{\sigma_{11}, \sigma_{22}, \sigma_{33}, \sqrt{2}\,\sigma_{12}, \sqrt{2}\,\sigma_{23}, \sqrt{2}\,\sigma_{13}\right\}$,

data$$ - parameters defining the BP yield surface: {pc,c,m,α,β,γ,M,tolerance}, see [2,3] (tolerance is used to regularize the J_2 invariant, see the code),

dorder$$ - integer specifying the desired order of the derivative to be computed:

= 0 - only the yield function F is computed,
= 1 - F and its first derivative are computed,
= 2 - F, first and second derivative are computed.

The output parameters are the following:

F$$ - yield function F,
DF$$ - first derivative $\partial F / \partial \mathbf{x}$,
DDF$$ - second derivative $\partial^2 F \,/\, \partial \mathbf{x}\, \partial \mathbf{x}$,
iter$$ - number of Newton iterations or -1 if no convergence is achieved,

where vector $\mathbf{x}$ comprises the stress components and yield surface parameters, namely $\mathbf{x} = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5, \sigma_6, \text{pc}, c, m, \alpha, \beta, \gamma, M\}$.

Ready-to-use generated codes in *Mathematica*, *Fortran*, *C* and *Matlab* are also provided on the web page (http://www.ippt.pan.pl/~sstupkie/files/BP_yield_function.html). The *Mathematica* code has been tested, and examples of its use are given in this notebook. The remaing codes have not been tested but they should work, as they have been generated from the same *AceGen* input.

Note that, to generate the code, you can use the trial version of *AceGen* which can be downloaded from

http://www.fgg.uni-lj.si/Symech/.

Feel free to use and modify this notebook for your work provided the source, i.e. our paper, is adequately cited.

■ Generate BPYieldFunction[]

Load *AceGen* packege and initialize *AceGen* session. The desired language (*Mathematica*, *Fortran*, *C* or *Matlab*) is selected as an option of SMSInitialize[ ].

In[1]:=
```
<< AceGen`;
```

In[2]:=
```
SMSInitialize[name = "BPYieldFunction", "Mode" → "Optimal",
  "Language" → {"Mathematica", "Fortran", "C", "Matlab"}[[1]]];
SMSModule[name
  , Real[sigma$$[6], data$$[8], F$$, DF$$[13], DDF$$[13, 13]]
  , Integer[dorder$$, iter$$]
  , "Input" → {sigma$$, data$$, dorder$$}
  , "Output" → {F$$, DF$$, DDF$$, iter$$}
 ];
```

Take the input parameters from the arguments of the subroutine:

In[4]:=
```
σ ⊨ SMSReal[Array[sigma$$[#1] &, 6], -0.01, 0.01];
{pc, c, m, α, β, γ, M, tolerance} ⊨ Array[SMSReal[data$$[#1]] &, 8];
dorder ⊨ SMSInteger[dorder$$];
```

Compute the stress invariants and auxiliary variables. J_2 is regularized to avoid division by zero for $q = 0$. Variables p, q and cos3θ are "frozen" (using SMSFreeze[ ]) so that differentiation with respect to them is possible.

In[7]:=
```
p ⊨ SMSFreeze[-(σ[[1]] + σ[[2]] + σ[[3]]) / 3];
σdev ⊨ σ + p {1, 1, 1, 0, 0, 0};
tol = tolerance pc;
J2 ⊨ 1 / 2 σdev.σdev + tol^2;
J3 ⊨ Det[σdev // {{#[[1]], #[[4]] / √2, #[[6]] / √2},
      {#[[4]] / √2, #[[2]], #[[5]] / √2}, {#[[6]] / √2, #[[5]] / √2, #[[3]]}} &];
q ⊨ SMSFreeze[SMSSqrt[3 J2]];
rh ⊨ (pc + c) / 2;
pr ⊨ -c + rh;
r ⊨ SMSSqrt[(p - pr)^2 + 3 J2];
cos3θ ⊨ SMSFreeze[3 √3 / 2 J3 / SMSSqrt[J2]^3];
g ⊨ 1 / Cos[β π / 6 - 1 / 3 ArcCos[γ cos3θ]];
ψ ⊨ ArcTan[p - pr, q];
```

```
Function ArcTan×1 is potentially unsafe. See also: Expression Optimization
Function ArcCos×1 is potentially unsafe. See also: Expression Optimization
Function ArcCos×1 is potentially unsafe. See also: Expression Optimization
Function ArcTan×1 is potentially unsafe. See also: Expression Optimization
```

Define function that computes the squared BP yield function $\hat{F}_0(\rho_0)$. Note that the squared version of the original yield function is used here, cf. Eq. (20).

In[19]:=
```
F0fun[r0a_] := Module[{},
  p0 ⊨ pr + r0a Cos[ψ];
  q0 ⊨ r0a Sin[ψ];
  ϕ ⊨ (p0 + c) / (pc + c);
  F0 ⊨ -M^2 pc^2 (ϕ - ϕ SMSPower[SMSAbs[ϕ], m - 1, "NonNegative"]) (2 (1 - α) ϕ + α) + (q0 / g)^2;
 ];
```

Define function that finds ρ_0 by solving nonlinear equation $\hat{F}_0(\rho_0) = 0$ using Newton method. Once the solution is found, the first derivative (Dr0Dσ) and the second derivative (D2r0Dσ2) are computed.

In[20]:=
```
r0Newton[r0ini_] := Module[{},
  r0i ⇇ SMSReal[r0ini];
  SMSDo[jNR, 1, 15, 1, r0i];
    F0fun[r0i];
    δFδr ⊨ SMSD[F0, r0i];
    δ2Fδr2 ⊨ SMSD[δFδr, r0i];
    Δr0 ⊨ -F0 / δFδr;
    r0i ↤ r0i + Δr0;
    SMSIf[Abs[Δr0] < 10^-10 && Abs[F0] < 10^-8];
      δFδσ = SMSD[F0, r0Implicit, "Constant" → r0i];
      Dr0Dσ ⇇ -δFδσ / δFδr;
      δ2Fδσ2 ⊨ SMSD[δFδσ, r0Implicit, "Constant" → r0i];
      δ2Fδrδσ ⊨ SMSD[δFδr, r0Implicit, "Constant" → r0i];
      aux1 = δ2Fδr2 Outer[Times, Dr0Dσ, Dr0Dσ];
      aux2 = Outer[Times, Dr0Dσ, δ2Fδrδσ];
      D2r0Dσ2 ⇇ -(δ2Fδσ2 + aux1 + aux2 + Transpose[aux2]) / δFδr;
      SMSExport[jNR, iter$$];
      SMSBreak[];
    SMSEndIf[];
    SMSIf[jNR == 15];
      SMSExport[-1, iter$$];
      SMSBreak[];
    SMSEndIf[];
  SMSEndDo[r0i, Dr0Dσ, D2r0Dσ2];
 ];
```

Solve nonlinear equation $\hat{F}_0(\rho_0) = 0$ using Newton method:

In[21]:=
```
r0Implicit = SMSVariables[{p, q, cos3θ, pc, c, m, α, β, γ, M}];
r0Newton[rh];
```

```
Function ArcCos×6 is potentially unsafe. See also: Expression Optimization
Function ArcCos×6 is potentially unsafe. See also: Expression Optimization
Function ArcCos×54 is potentially unsafe. See also: Expression Optimization
Function ArcCos×6 is potentially unsafe. See also: Expression Optimization
```

Introduce AD exceptions that define the second and first derivative of the solution with respect to implicit variables stored in r0Implicit:

In[23]:=
```
δr0 ⊨ SMSFreeze[Dr0Dσ, "Dependency" → {r0Implicit, D2r0Dσ2}];
r0 ⊨ SMSFreeze[r0i, "Dependency" → Transpose[{r0Implicit, δr0}]];
```

Compute and export the value of the (implicit) yield function:

In[25]:=
```
F ⊨ r / r0 - 1;
SMSExport[F, F$$];
```

Compute and export the first and second derivatives of F with respect to $\mathbf{x} = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5, \sigma_6, \text{pc}, c, m, \alpha, \beta, \gamma, M\}$:

In[27]:=
```
SMSIf[dorder ≥ 1];
    DF ⊨ SMSD[F, Flatten[{σ, {pc, c, m, α, β, γ, M}}]];
    SMSExport[DF, DF$$];
    SMSIf[dorder == 2];
        DDF ⊨ SMSD[DF, Flatten[{σ, {pc, c, m, α, β, γ, M}}]];
        SMSExport[DDF, DDF$$];
    SMSElse[];
        SMSExport[Table[0, {13}, {13}], DDF$$];
    SMSEndIf[];
SMSElse[];
    SMSExport[Table[0, {13}], DF$$];
SMSEndIf[];
```

Generate source code:

In[39]:=
```
SMSWrite[];
```

File:	BPYieldFunction.m	Size:	54 023
Methods	No.Formulae	No.Leafs	
BPYieldFunction	874	15 009	

■ Test the *Mathematica* code

This cell shows how to use the *Mathematica* code.

In[40]:=
```
SetDirectory[NotebookDirectory[]];
```

In[41]:=
```
<< "BPYieldFunction.m"
```

In[42]:=
```
(* {pc,c,m,α,β,γ,M,tolerance} *)
data = {10., 0.5, 2., 0.1, 0.19, 0.9, 1.1, 10.^-6};
```

In[75]:=
```
SeedRandom[1];
σ = RandomReal[{-1, 1}, 6]
```

Out[76]=
```
{0.634779, -0.777161, 0.579052, -0.624394, -0.517278, -0.868522}
```

In[77]:=
```
DF = Table[0., {13}];
DDF = Table[0., {13}, {13}];
```

In[79]:=
```
dorder = 0;
BPYieldFunction[σ, data, F, DF, DDF, dorder, iter];
{F, iter}
```

Out[81]=
```
{0.124351, 5}
```

In[82]:=
```
dorder = 1;
BPYieldFunction[σ, data, F, DF, DDF, dorder, iter];
{F, DF, iter}
```

Out[84]=
```
{0.124351, {0.105474, -0.0163746, 0.0914432, -0.0230308, -0.00826285, -0.132974,
  -0.017955, -0.174448, -0.0276081, -0.39328, 0.0726822, -0.015797, -0.218632}, 5}
```

In[85]:=
```
order = 2;
BPYieldFunction[σ, data, F, DF, DDF, order, iter];
{F, DF, DDF, iter}
```

Out[87]=
```
{0.124351, {0.105474, -0.0163746, 0.0914432, -0.0230308, -0.00826285, -0.132974,
  -0.017955, -0.174448, -0.0276081, -0.39328, 0.0726822, -0.015797, -0.218632},
 {{0.0865324, -0.00443777, -0.0828348, -0.0192731, 0.0174647, 0.0113945, -0.0106481,
   -0.00501635, -0.00798636, -0.0401038, 0.0128942, -0.000902813, -0.0444696},
  {-0.00443777, 0.0142215, -0.00740527, -0.00820817, -0.00112967, -0.0013242,
   0.00316873, -0.00802972, 0.017744, 0.0160455, -0.0338211, 0.0145557, 0.0801814},
  {-0.0828348, -0.00740527, 0.089859, 0.0280707, -0.0161236, -0.00666692, -0.00905714,
   -0.00536334, -0.00502351, -0.0336383, 0.0190666, -0.0132485, -0.0301161},
  {-0.0192731, -0.00820817, 0.0280707, 0.0732973, -0.0468535, -0.00959198, 0.00261155,
   -0.000569561, 0.00486334, 0.0106129, -0.0474788, 0.0501836, 0.0235605}, {0.0174647,
   -0.00112967, -0.0161236, -0.0468535, 0.0641031, -0.000312906, 0.000936956,
   -0.000204344, 0.00174484, 0.00380762, -0.0487367, 0.0567717, 0.00845289},
  {0.0113945, -0.0013242, -0.00666692, -0.00959198, -0.000312906, 0.0307634,
   0.0150784, -0.0032885, 0.0280797, 0.0612759, 0.0217006, -0.0720078, 0.136032},
  {-0.0106481, 0.00316873, -0.00905714, 0.00261155, 0.000936956, 0.0150784,
   0.00399337, 0.0153979, 0.00523826, 0.00913465, -0.00824171, 0.00179128, 0.0247915},
  {-0.00501635, -0.00802972, -0.00536334, -0.000569561, -0.000204344, -0.0032885,
   0.0153979, 0.0344021, -0.00457436, 0.055749, 0.00179746, -0.000390666, -0.00540685},
  {-0.00798636, 0.017744, -0.00502351, 0.00486334, 0.00174484, 0.0280797,
   0.00523826, -0.00457436, 0.0823846, 0.0449655, -0.0153481, 0.0033358, 0.0461678},
  {-0.0401038, 0.0160455, -0.0336383, 0.0106129, 0.00380762, 0.0612759,
   0.00913465, 0.055749, 0.0449655, 0.925002, -0.0334928, 0.00727944, 0.100748},
  {0.0128942, -0.0338211, 0.0190666, -0.0474788, -0.0487367, 0.0217006, -0.00824171,
   0.00179746, -0.0153481, -0.0334928, -0.0631807, 0.0137319, -0.0743538},
  {-0.000902813, 0.0145557, -0.0132485, 0.0501836, 0.0567717, -0.0720078,
   0.00179128, -0.000390666, 0.0033358, 0.00727944, 0.0137319, -0.00464163, 0.0161603},
  {-0.0444696, 0.0801814, -0.0301161, 0.0235605, 0.00845289, 0.136032, 0.0247915,
   -0.00540685, 0.0461678, 0.100748, -0.0743538, 0.0161603, 0.422416}}, 5}
```

■ Check first and second derivatives

In this cell, the derivatives are checked with respect to finite difference approximation. For the finite-difference perturbation between $\Delta = 10^{-7}$ and $\Delta = 10^{-9}$ the relative error is of the order of 10^{-7} which verifies that the derivatives are computed correctly.

In[88]:=
```
SetDirectory[NotebookDirectory[]];
```

In[89]:=
```
<< "BPYieldFunction.m"
```

In[90]:=
```
(* {pc,c,m,α,β,γ,M,tolerance} *)
data = {10., 0.5, 2., 0.1, 0.19, 0.9, 1.1, 10.^-6};
```

In[91]:=
```
SeedRandom[2];
σ = RandomReal[{-10, 10}, 6]
```

Out[92]=
```
{4.4448, -7.81103, -0.585946, 0.711637, 1.66355, -4.12115}
```

In[93]:=
```
DF = Table[0., {13}];
DDF = Table[0., {13}, {13}];
```

In[95]:=
```
input = Join[σ, data[[1 ;; -2]]]
```

Out[95]=
```
{4.4448, -7.81103, -0.585946, 0.711637,
 1.66355, -4.12115, 10., 0.5, 2., 0.1, 0.19, 0.9, 1.1}
```

This checks the first derivative:

In[98]:=
```
Table[
 Δ = 10.^-iΔ;
 BPYieldFunction[input[[1 ;; 6]], Append[input[[7 ;; 13]], data[[8]]], F, DF, DDF, 2, iter];
 res0 = {F, DF, DDF, iter};
 DFnum = Table[
   inputP = input;
   inputP[[iinp]] = input[[iinp]] + Δ;
   BPYieldFunction[inputP[[1 ;; 6]],
    Append[inputP[[7 ;; 13]], data[[8]]], F, DF, DDF, 1, iter];
   resP = {F, DF, DDF, iter};
   (resP[[1]] - res0[[1]]) / Δ
   , {iinp, Length[input]}];
 Norm[res0[[2]] - DFnum] / Norm[res0[[2]]]
 , {iΔ, 4, 10}]
```

Out[98]=
$$\left\{0.0000902006,\ 9.02081\times10^{-6},\ 9.02349\times10^{-7},\ 8.9777\times10^{-8},\ 1.02543\times10^{-7},\ 7.02175\times10^{-7},\ 8.09165\times10^{-6}\right\}$$

This checks both the first and the second derivative:

In[99]:=
```
Table[
 Δ = 10.^-iΔ;
 BPYieldFunction[input[[1 ;; 6]], Append[input[[7 ;; 13]], data[[8]]], F, DF, DDF, 2, iter];
 res0 = {F, DF, DDF, iter};
 Dnum = Table[
   inputP = input;
   inputP[[iinp]] = input[[iinp]] + Δ;
   BPYieldFunction[inputP[[1 ;; 6]],
    Append[inputP[[7 ;; 13]], data[[8]]], F, DF, DDF, 1, iter];
   resP = {F, DF, DDF, iter};
   DFn = (resP[[1]] - res0[[1]]) / Δ;
   DDFn = (resP[[2]] - res0[[2]]) / Δ;
   {DFn, DDFn}
   , {iinp, Length[input]}];
 {Norm[res0[[2]] - Dnum[[All, 1]]] / Norm[res0[[2]]],
  Norm[res0[[3]] - Dnum[[All, 2]]] / Norm[res0[[3]]]}
 , {iΔ, 4, 10}]
```

Out[99]=
$$\left\{\{0.0000902006,\ 0.000132043\},\ \left\{9.02081\times10^{-6},\ 0.0000132057\right\},\ \left\{9.02349\times10^{-7},\ 1.32059\times10^{-6}\right\},\ \left\{8.9777\times10^{-8},\ 1.30364\times10^{-7}\right\},\ \left\{1.02543\times10^{-7},\ 4.88653\times10^{-8}\right\},\ \left\{7.02175\times10^{-7},\ 4.40748\times10^{-7}\right\},\ \left\{8.09165\times10^{-6},\ 5.61609\times10^{-6}\right\}\right\}$$

■ Contour plot in (σ_1, σ_2)-plane

In[100]:=
```
SetDirectory[NotebookDirectory[]];
```

In[2]:=
```
<< "BPYieldFunction.m"
```

In[101]:=
```
(* {pc,c,m,α,β,γ,M,tolerance} *)
data = {10., 0.5, 2., 0.1, 0.19, 0.9, 1.1, 10.^-6};
```

In[102]:=
```
DF = Table[0., {13}];
DDF = Table[0., {13}, {13}];
```

In[104]:=
```
{σmin, σmax} = {-5., 5.};
list =
  Table[{σ1, σ2, (BPYieldFunction[{σ1, σ2, 0., 0., 0., 0.}, data, F, DF, DDF, 0, iter]; F)}
   , {σ1, σmin, σmax, (σmax - σmin) / 20}, {σ2, σmin, σmax, (σmax - σmin) / 20}];
```

In[106]:=
```
list // Flatten[#, 1] & // ListContourPlot
```

Out[106]=