In[73]:=

```
(* Initialisation *)
(* Evaluate before starting writing "real code" *)
(* Usage e.g.: "ld [Spacekey]" becomes "⊨",
so writing "a ld 5" turns into "a⊨5" *)
SetOptions[EvaluationNotebook[],
            InputAutoReplacements → {(* special AceGen assignment operators: *)
   "ld" → "⊨", "ls" → "⊢", "rd" → "⫤", "rs" → "⊣",
                                    (* brackets and symbols: *) "dbl" → "⟦",
   "dbr" → "⟧", "lcb" → "{", "rcb" → "}", "lsb" → "[", "rsb" → "]", "->" → "→",
                                    (* shortcuts for
    starting/ending a comment block: *) "co" → "(*", "cc" → "*)"
                                   }
          ]
(* Output the current time,
so we know when AceGen has been executed the last time *)
Now
```

Out[74]=

```
Thu 1 Aug 2024 15:58:07 GMT+2
```

In[75]:=
```
(* Clear all old variables initially to have a fresh start *)
ClearAll["Global`*"]
(* Start AceGen *)
<< AceGen`;
(* Load the AceTensorFunctions package that
 contains helpful functions for operating with tensors *)
(* @note Sometimes Mathematica fails to load such
 packages. Try rerunning the notebook, restart the kernel,
retype the backtick ` at the end of AceTensorFunctions, etc. *)
Needs["AceTensorFunctions` "];
(* Name of the to be created subroutine/function
 in the below specified programming language *)
NAME = "LinearElasticity";
(* Name of the AceGen session "NAME",
specify the programming language "Language"={C++,Matlab,Fortran,...},
and the execution mode "Mode"={Optimal,Prototype,Debug,Plain} *)
(* @note Changing the output programming language can be very simple here,
so feel free to take advantage of all the available languages. For instance,
first export to a Matlab-code,
because you can quickly and easily debug and check the code and its output. *)
SMSInitialize[NAME, "Language" → "Matlab", "Mode" → "Optimal"];
(* Start a module, which represents the to be created function,
with name "NAME" and the specified input and output arguments *)
SMSModule[NAME, Real[deformationGradient$$[3, 3],
   listOfMaterialParameters$$[2], CauchyStressVEC$$[6], Tangent$$[3, 3, 3, 3]],
          "Input" → {deformationGradient$$, listOfMaterialParameters$$},
          "Output" → {CauchyStressVEC$$, Tangent$$}
         ];
(* Input declaration by copying AceGen variables to Mathematica variables *)
deformationGradient ⊨ SMSReal[Table[deformationGradient$$[i, j], {i, 3}, {j, 3}]];
listOfMaterialParameters ⊨ SMSReal[Table[listOfMaterialParameters$$[i], {i, 2}]];
```

In[83]:=
```
(* Extract the bulk modulus kappa and the shear
 modulus mu from the list of material parameters *)
(* @note You cannot use the default underscore "_" in variable names,
instead the special character "[Esc]_[Esc]" is used. *)
(* @note You can also use greek/etc. symbols like "α" for variables as
 in classical Mathematica notebooks, e.g. "[Esc]kappa[Esc]" for κ. *)
bulkMod_kappa ⊨ listOfMaterialParameters [[1]];
shearMod_mu ⊨ listOfMaterialParameters [[2]];
(* Compute the geometrically linear strain tensor
 "glStrain_H" from the deformation gradient and "freeze" it,
because we later take the derivative with respect to this variable *)
(* @note If you directly take derivatives with
 respect to an input argument like "deformationGradient",
the "freeze" is not required, because "deformationGradient" already has a a so-
 called "unique signature" by using SMSReal *)
SMSFreeze[glStrain_H , 1/2 (deformationGradient - IdentityMatrix[3] +
     Transpose[deformationGradient - IdentityMatrix[3]]), "Symmetric" → True];
(* Compute the stress tensor for linear elasticity based on the
 geometrically linear strain and the material parameters *)
(* @note Note the use of the function "devten" from AceTensorFunction
 to compute the deviatoric part of the tensor "glStrain_H" *)
stress ⊨ bulkMod_kappa * Tr[glStrain_H] * IdentityMatrix[3] +
   2 * shearMod_mu * devten[glStrain_H];
```

In[87]:=
```
(* Export the output variables by copying
 the Mathematica variables to AceGen variables *)
(* Transform the second order stress tensor "stress" into Voigt vector
 notation (6 component vector, here ordered as xx,yy,zz,xy,yz,zx) *)
stressVEC ⊨ ten2vec[stress, 1];
(* Output/Export the stress vector "stressVEC" as variable CauchyStressVEC$$ *)
SMSExport[stressVEC , Table[CauchyStressVEC$$[i], {i, 6}]];

(* Compute the derivative of the stress tensor
 with respect to the geometrically linear strain *)
Dstress_DglStrain ⊨ SMSD[stress, glStrain_H, "Symmetric" → True];
(* Output/Export the derivative as a fourth-order tensor *)
(* @note Symmetry of the derivative is numerically not ensured,
therefore it is recommended to either use the option
 "Symmetric" or take derivatives in Voigt/Nye/Vector notation *)
SMSExport[Dstress_DglStrain , Table[Tangent$$[i, j, k, l], {i, 3}, {j, 3}, {k, 3}, {l, 3}]];
```

In[91]:=
```
(* Debugging *)
(* Output the stress tensor to the screen *)
SMSPrintMessage[NAME <> "<< stressVEC=", stressVEC];
(* Compute the analytical tangent and compare it to the AceGen-Output *)
(* @note Note that for this simple model,
Mathematica sometimes even optimises the expression
 for the error shown below in the generated code *)
tangent_analytical ⊨ bulkMod_kappa *
    Table[KroneckerDelta[i, j] * KroneckerDelta[k, l], {i, 3}, {j, 3}, {k, 3}, {l, 3}] +
   2 * shearMod_mu * Table[-1/3 * KroneckerDelta[i, j] * KroneckerDelta[k, l] +
      1/2 * KroneckerDelta[i, k] * KroneckerDelta[j, l] +
      1/2 * KroneckerDelta[i, l] * KroneckerDelta[j, k], {i, 3}, {j, 3}, {k, 3}, {l, 3}];
SMSPrintMessage[NAME <> "<< error in tangent=",
  Sum[(Dstress_DglStrain - tangent_analytical)〚i, j, k, l〛, {i, 3}, {j, 3}, {k, 3}, {l, 3}]];
```

In[94]:=
```
(* Output the time at the end of the execution *)
Now
(* Write output file containing all the
 above defined functions introduced by SMSModule *)
(* Create output file named "NAME", '"LocalAuxiliaryVariables" →
 True' is a command to exclude the AceGen internal array "v" from
  the list of input and output arguments of the created subroutine *)
SMSWrite[NAME, "LocalAuxiliaryVariables" → True];
(* Print the content of the just created
 file on screen (sensible only for small file sizes) *)
FilePrint[StringJoin[NAME, Which[SMSLanguage == "Fortran", ".f",
                              SMSLanguage == "Matlab", ".m",
                              SMSLanguage == "C++", ".cpp",
                              SMSLanguage == "C", ".c"
                             ]
                   ]
         ]
```

Out[94]= Thu 1 Aug 2024 15:58:07 GMT+2

File: LinearElasticity.m **Size:** 3802 **Time:** 1

Method	LinearElasticity
No.Formulae	21
No.Leafs	1026

```
%*************************************************************
%* AceGen    7.505 Linux (16 Aug 22)                         *
%*           Co. J. Korelc  2020           1 Aug 24 15:58:08  *
%*************************************************************
% User      : Full professional version
% Notebook  : AceGen-LinearElasticity
```

```
% Evaluation  time                   : 1 s      Mode   : Optimal
% Number  of  formulae               : 21       Method:  Automatic
% Subroutine                         : LinearElasticity  size:  1026
% Total  size  of  Mathematica   code : 1026  subexpressions
% Total  size  of  Matlab  code      : 2889  bytes

%*********************   F U N C T I O N *********************
function[CauchyStressVEC,Tangent]=LinearElasticity(deformationGradient,listOfMaterialParamet
persistent  v;
if size(v)<144
  v=zeros(144,'double');
end;
v(10)=listOfMaterialParameters(1);
v(11)=listOfMaterialParameters(2);
v(34)=2e0*v(11);
v(29)=v(10)+(-2e0/3e0)*v(11);
v(28)=v(10)+(4e0/3e0)*v(11);
v(12)=-1e0+deformationGradient(1,1);
v(15)=-1e0+deformationGradient(2,2);
v(17)=-1e0+deformationGradient(3,3);
v(18)=v(12)+v(15)+v(17);
v(23)=v(10)*v(18);
v(22)=(-1e0/3e0)*v(18);
v(19)=v(23)+(v(12)+v(22))*v(34);
v(20)=(deformationGradient(1,2)+deformationGradient(2,1))*v(11);
v(21)=(deformationGradient(1,3)+deformationGradient(3,1))*v(11);
v(24)=v(23)+(v(15)+v(22))*v(34);
v(25)=(deformationGradient(2,3)+deformationGradient(3,2))*v(11);
v(26)=v(23)+(v(17)+v(22))*v(34);
CauchyStressVEC(1)=v(19);
CauchyStressVEC(2)=v(24);
CauchyStressVEC(3)=v(26);
CauchyStressVEC(4)=v(20);
CauchyStressVEC(5)=v(25);
CauchyStressVEC(6)=v(21);
Tangent(1,1,1,1)=v(28);
Tangent(1,1,1,2)=0;
Tangent(1,1,1,3)=0;
Tangent(1,1,2,1)=0;
Tangent(1,1,2,2)=v(29);
Tangent(1,1,2,3)=0;
Tangent(1,1,3,1)=0;
Tangent(1,1,3,2)=0;
Tangent(1,1,3,3)=v(29);
Tangent(1,2,1,1)=0;
Tangent(1,2,1,2)=v(11);
Tangent(1,2,1,3)=0;
Tangent(1,2,2,1)=v(11);
Tangent(1,2,2,2)=0;
Tangent(1,2,2,3)=0;
Tangent(1,2,3,1)=0;
Tangent(1,2,3,2)=0;
Tangent(1,2,3,3)=0;
Tangent(1,3,1,1)=0;
Tangent(1,3,1,2)=0;
```

```
Tangent(1,3,1,3)=v(11);
Tangent(1,3,2,1)=0;
Tangent(1,3,2,2)=0;
Tangent(1,3,2,3)=0;
Tangent(1,3,3,1)=v(11);
Tangent(1,3,3,2)=0;
Tangent(1,3,3,3)=0;
Tangent(2,1,1,1)=0;
Tangent(2,1,1,2)=v(11);
Tangent(2,1,1,3)=0;
Tangent(2,1,2,1)=v(11);
Tangent(2,1,2,2)=0;
Tangent(2,1,2,3)=0;
Tangent(2,1,3,1)=0;
Tangent(2,1,3,2)=0;
Tangent(2,1,3,3)=0;
Tangent(2,2,1,1)=v(29);
Tangent(2,2,1,2)=0;
Tangent(2,2,1,3)=0;
Tangent(2,2,2,1)=0;
Tangent(2,2,2,2)=v(28);
Tangent(2,2,2,3)=0;
Tangent(2,2,3,1)=0;
Tangent(2,2,3,2)=0;
Tangent(2,2,3,3)=v(29);
Tangent(2,3,1,1)=0;
Tangent(2,3,1,2)=0;
Tangent(2,3,1,3)=0;
Tangent(2,3,2,1)=0;
Tangent(2,3,2,2)=0;
Tangent(2,3,2,3)=v(11);
Tangent(2,3,3,1)=0;
Tangent(2,3,3,2)=v(11);
Tangent(2,3,3,3)=0;
Tangent(3,1,1,1)=0;
Tangent(3,1,1,2)=0;
Tangent(3,1,1,3)=v(11);
Tangent(3,1,2,1)=0;
Tangent(3,1,2,2)=0;
Tangent(3,1,2,3)=0;
Tangent(3,1,3,1)=v(11);
Tangent(3,1,3,2)=0;
Tangent(3,1,3,3)=0;
Tangent(3,2,1,1)=0;
Tangent(3,2,1,2)=0;
Tangent(3,2,1,3)=0;
Tangent(3,2,2,1)=0;
Tangent(3,2,2,2)=0;
Tangent(3,2,2,3)=v(11);
Tangent(3,2,3,1)=0;
Tangent(3,2,3,2)=v(11);
Tangent(3,2,3,3)=0;
Tangent(3,3,1,1)=v(29);
Tangent(3,3,1,2)=0;
Tangent(3,3,1,3)=0;
```

```
Tangent(3,3,2,1)=0;
Tangent(3,3,2,2)=v(29);
Tangent(3,3,2,3)=0;
Tangent(3,3,3,1)=0;
Tangent(3,3,3,2)=0;
Tangent(3,3,3,3)=v(28);
disp(sprintf("\n%s  %f %f %f %f %f %f ","LinearElasticity<<  stressVEC=",v(19),v(24),v(26),v(20),
 ),v(21)));
disp(sprintf("\n%s  %f ","LinearElasticity<<  error in tangent=",0));


function [x]=SMSKDelta(i,j)

if (i==j) , x=1; else x=0; end;

end

function [x]=SMSDeltaPart(a,i,j,k)

l=round(i/j);

if (mod(i,j) ~= 0 | l>k) , x=0; else x=a(l); end;

end

function [x]=Power(a,b)

x=a^b;

end

function [x]=SMSTernaryOperator(a,b,c)

if (c) , x=a; else x=b; end;

end

end
```