Symbolic manipulation of layered media

Dave Nichols and Rick Ottolini

ABSTRACT

Fundamental properties of elastic media based on the Schoenberg/Muir calculus can be studied using the symbolic mathematics program Mathematica.

INTRODUCTION

Schoenberg and Muir (1988) described a group-theoretic method for manipulating the elastic properties of layered and fractured media. Nichols (1988) presented a computational toolkit that implemented this theory. This gave numerical results from the manipulation of model layered media. However there are still implications of the theory that cannot be investigated using numerical examples. The tedious algebra required to prove any general results using the Schoenberg/Muir method has prevented much progress in this direction.

The symbolic mathematics program *Mathematica*^{TM} (Wolfram, 1988) was recently released for the Macintosh computer it is described as "A system for doing mathematics by computer". We have implemented Nichols (1988) layer modeling toolkit in Mathematica. It can be used to manipulate symbolic equations that would be difficult to do by hand.

We first introduce Mathematica. Next we show some examples of the implementation of the Schoenberg/Muir calculus. Finally we show some basic results about the properties of elastic media and suggest some possible further investigations.

A BRIEF INTRODUCTION TO MATHEMATICA

Mathematica is a very flexible program and can be used in many ways. At its simplest it can be used as a calculator, the user enters calculations using a text interface, if you enter **2 + 2** you get the response **4**. However you are not limited

to entering numbers, the operands in any calculation can be symbols as well. If you enter **c = a + b** you get the response **a + b**, if you then enter **d= 2 c** you get the response **2 a + 2 b**. Mathematica has remembered the definition of **c** and used it to define **d**.

You can also type calculations using one of the many predefined functions. If you type **Sin[Pi/6]** you get the response **0.5**. Mathematica also supports calculation on arrays, arbitrary precision arithmetic, 3-D plots of calculation results, and most importantly it allows you to define your own transformation rules and functions.

Mathematica has many built in rules for solving equations and predefined functions. It has a function **TrigReduce[]** that attempts to reduce the complexity of an expression by applying rules about trigonometric functions. One of the rules that is used is **Sin[x_ + y_] :> Sin[x] Cos[y] + Cos[x] Sin[y]** . When this rule is applied any expression of the form on the left hand side is converted to the form on the right hand side. The terms **x_** and **y_** can be anything, a number, a polynomial, another function etc. In general Mathematica operates by matching patterns within expressions so the rule shown matches a **Sin[]** where the operand is two expression separated by a plus sign. Other functions in Mathematica can be used to factor or expand polynomials to simplify expressions, differentiate etc.

A program in Mathematica is a set of rules. When you type in an expression Mathematica goes through the rules applying all those that are valid. When nothing further can be changed the "program" is finished. It is possible to get caught in a loop in Mathematica so it checks whether you are repeating the same functions with the same arguments and stops if you do this too many times.

Specifying rules in Mathematica

Mathematica stores sets of rules and functions related to a particular topic in a "notebook". In order to use the rules in a notebook you just read in the notebook before starting your session. Notebooks allow you to write your function definitions and documentation in "cells" you can then group your cells into sensible sections and subsections. If you choose you can hide all the cells containing function definitions so that only the documentation shows. Appendix II is a listing of the Schoenberg/Muir notebook documentation and two of the definition cells shown.

To specify your own functions you need to specify a pattern to be matched and a function that is evaluated when that pattern is found. If you specify

f[x_] := (2 x + 1)

then all any time you evaluate an equation containing

f[some expression]

it will be replaced by

(2 * your expression + 1).

This method of evaluating expressions is very flexible and you can write Mathematica programs in many different styles. In the Schoenberg/Muir notebook we have chosen to write the rules in a style similar to object oriented programming. We define rules that apply only to a certain type of object. We define one type of object, a layer, as being a list of five objects :

1. The layer thickness
2. the layer density
3. The C_{nn} submatrix
4. The C_{nt} submatrix
5. The C_{tt} submatrix

Appendix II contains a brief refresher course on the Schoenberg/Muir calculus and definitions of the symbols used.

In our notebook a layer has the form : **Layer[list of components]**. The function to rotate a layer about the z-axis is then defined as :

```
Layer\: RotateZ[ Layer[ x_ ] , angle_ ] := { function definition }
```

The first "*Layer\ :*" defines this function as belonging to the group of functions dealing with layers. The pattern to be matched is a call to RotateZ where the first argument is a layer and the second argument is anything. This function definition will not be applied unless you give it a layer as an argument. You can supply another definition of the function that is valid when some other type of object is supplied as an argument.

One of the nice features of Mathematica is that you can redefine the standard mathematical operators. The function to add two objects has the form **Plus[a_, b_]**. If you type **2+2** Mathematica internally stores it as **Plus[2 , 2]**. We have redefined the Plus function for layers so that when two layers are added they are added in the in the Schoenberg/Muir sense. If you type **a+b** and both **a** and **b** are layers our new definition of add will be used. The definition looks like this:

```
Layer\: Plus[ Layer[a_], Layer[b_] ] :=
    GtoL[
         LtoG[ Layer[a] ]  +  LtoG[ Layer[b] ]
        ]
```

This should be interpreted in the following way, if asked to add two layers, first map them to the group domain, then add the two groups, take the result and map back to the layer domain. When evaluating expressions Mathematica first looks for a specific definition of each function if it doesn't find one it looks for a general definition of the function. In this way our definition of addition overrides the default definition that would just have added corresponding elements in the list.

EXAMPLES OF SYMBOLIC CALCULATIONS

Adding layers

The commands :

a = MakeLayer[z1, rho1, LameMatrix[lam1, mu1, mu1]] ,

b = MakeLayer[z2, rho2, LameMatrix[lam2, mu2, mu2]] ,

create two isotropic layers. They are combined using the command :

c = a + b ,

this produces a complicated expression which is simplified using the command:

d = Simplify[c].

We can test to see if the new layer is transverse isotropic using the command :

IsTI[d].

However this gives a long symbolic result so we use the **Reduce[]** command to reduce the result to its simplest form.

Reduce[IsTI[d]] ,

gives the result **True** so we have shown that the result of combining any two sets of isotropic layers is a medium with at least transverse isotropic symmetry.

The printout of the layers can be formated for TEX output using the command **TeXForm[]** . The output is often not immediately usable in a TEX document like this one but it takes very little effort to edit it to a suitable form. The first two layers, **a** and **b**, have the following layer properties,

layer a

$thickness = z_1$

$density = \rho_1$

$$\begin{pmatrix} \lambda_1+2\mu_1 & \lambda_1 & \lambda_1 & 0 & 0 & 0 \\ \lambda_1 & \lambda_1+2\mu_1 & \lambda_1 & 0 & 0 & 0 \\ \lambda_1 & \lambda_1 & \lambda_1+2\mu_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu_1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mu_1 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mu_1 \end{pmatrix}$$

layer b

$thickness = z_2$

$density = \rho_2$

$$\begin{pmatrix} \lambda_2+2\mu_2 & \lambda_2 & \lambda_2 & 0 & 0 & 0 \\ \lambda_2 & \lambda_2+2\mu_2 & \lambda_2 & 0 & 0 & 0 \\ \lambda_2 & \lambda_2 & \lambda_2+2\mu_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mu_2 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mu_2 \end{pmatrix}$$

The combined layer, **d**, has a very complicated elastic matrix, printed below are the formulae for the thickness and density and the upper left term of the elastic matrix.

layer d

$thickness = z_1 + z_2$

$density = \frac{\rho_1 z_1 + \rho_2 z_2}{z_1 + z_2}$

$$\frac{4\mu_1\mu_2{z_1}^2+4{\mu_1}^2 z_1 z_2+4{\mu_2}^2 z_1 z_2+4\mu_1\mu_2{z_2}^2+\lambda_1\lambda_2(z_1+z_2)^2+\lambda_2\left(2\mu_1{z_1}^2+4\mu_2 z_1 z_2+2\mu_1{z_2}^2\right)+\lambda_1\left(2\mu_2{z_1}^2+4\mu_1 z_1 z_2+2\mu_2{z_2}^2\right)}{(z_1+z_2)(\lambda_2 z_1+2\mu_2 z_1+\lambda_1 z_2+2\mu_1 z_2)}$$

If we assume the two sets of component layers have the same thickness we can set both z_1 and z_2 to one. We do this using the commands,

$$\mathbf{z1 = 1} \ ,$$

$$\mathbf{z2 = 1} \ .$$

This simplifies the elastic matrix for layer **d** to ,

layer d

$thickness = 2$

$density = \frac{\rho_1+\rho_2}{2}$

$$\begin{pmatrix} \frac{2(\lambda_1+\mu_1+\mu_2)(\lambda_2+\mu_1+\mu_2)}{\lambda_1+\lambda_2+2\mu_1+2\mu_2} & \frac{2\lambda_1\lambda_2+\lambda_1\mu_1+\lambda_2\mu_1+\lambda_1\mu_2+\lambda_2\mu_2}{\lambda_1+\lambda_2+2\mu_1+2\mu_2} & \frac{2(\lambda_1\lambda_2+\lambda_2\mu_1+\lambda_1\mu_2)}{\lambda_1+\lambda_2+2\mu_1+2\mu_2} & 0 & 0 & 0 \\ \frac{2\lambda_1\lambda_2+\lambda_1\mu_1+\lambda_2\mu_1+\lambda_1\mu_2+\lambda_2\mu_2}{\lambda_1+\lambda_2+2\mu_1+2\mu_2} & \frac{2(\lambda_1+\mu_1+\mu_2)(\lambda_2+\mu_1+\mu_2)}{\lambda_1+\lambda_2+2\mu_1+2\mu_2} & \frac{2(\lambda_1\lambda_2+\lambda_2\mu_1+\lambda_1\mu_2)}{\lambda_1+\lambda_2+2\mu_1+2\mu_2} & 0 & 0 & 0 \\ \frac{2(\lambda_1\lambda_2+\lambda_2\mu_1+\lambda_1\mu_2)}{\lambda_1+\lambda_2+2\mu_1+2\mu_2} & \frac{2(\lambda_1\lambda_2+\lambda_2\mu_1+\lambda_1\mu_2)}{\lambda_1+\lambda_2+2\mu_1+2\mu_2} & \frac{2(\lambda_1+2\mu_1)(\lambda_2+2\mu_2)}{\lambda_1+\lambda_2+2\mu_1+2\mu_2} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{2\mu_1\mu_2}{\mu_1+\mu_2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{2\mu_1\mu_2}{\mu_1+\mu_2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{\mu_1+\mu_2}{2} \end{pmatrix}$$

The symmetry property of transverse isotropy

The command :

$$\mathbf{a = GenTI[\ c\ ],}$$

Creates a generic transverse isotropic elastic matrix using the symbol "c" for the independent constants (i.e. $c_{1,2}$, $c_{6,6}$, $c_{1,3}$, $c_{4,4}$, $c_{6,6}$). The layer is rotated "th" degrees about the Z-axis, the axis of symmetry, using the command :

$$\mathbf{b = RotateZ[\ a,\ th\ ]}$$

Again we get a result that needs simplification, we use two commands :

$$\mathbf{d = Simplify[\ b\ ],}$$

$$\mathbf{e = TrigReduce[\ d\ ],}$$

to simplify the result. The first command performs algebraic simplification, the second simplifies trigonometric terms. We get a result that is the same as our starting elastic matrix. We can check this using the command :

$$\mathbf{e == a\ ,}$$

which returns the result **True** . This shows that a transverse isotropic elastic matrix is invariant under rotation about the Z-axis.

Fracturing a layer

In this example we take a generic transverse isotropic layer and fracture it with vertical cracks in the plane of the X-axis. This is done by rotating the elastic constants 90 degrees about the X-axis, fracturing the layer horizontally, and rotating the elastic constants back to the original frame. The fracturing is done using the Schoenberg/Muir model, a fraction "ex" of the third group element of the second layer is added to the third group element of the first layer. The commands :

```
a = MakeLayer[ z, rho, GenTI[c] ]

b = RotateX[ a, 90 ]

d = SMCrack[ b, b, ex ]

e = Simplify[d]

f = RotateX[ e, -90 ]
```

give the resultant layer, **f** . We can first check if the layer is transverse isotropic. The command :

```
Reduce[ IsTI[ f ] ] ,
```

gives the result **==0 || ex == 0** . This means that the layer is transverse isotropic if all the coefficients are zero or the parameter **ex** which controls the amount of fracturing is zero. The transverse isotropic layer retains its symmetry only if there is zero fracturing. We can also test to see if the layer has orthorhombic symmetry :

```
Reduce[ IsOrth[ f ] ].
```

This gives the result **True** , showing that any transverse isotropic layer with any amount of vertical fracturing (in the Schoenberg/Muir sense) has orthorhombic symmetry.

LIMITATIONS

We are currently running Mathematica on a Macintosh II computer with 5 megabytes of memory. If we try to do more complicated symbolic calculations than those shown here we hit the memory limit on our machine. Mathematica will

be released soon in versions for other computers, the Sun workstation version has already been demonstrated. When the Sun version becomes available we will not have memory problems because the Sun has a virtual memory operating system. Our only limitation will be the length of time we are prepared to wait to get results.

CONCLUSIONS

Mathematica is a very flexible symbolic mathematics program that has enabled us to prove some well known results about anisotropic media. We cannot continue this work to prove new results until we have more processing power and memory available. When we are able to continue we intend to prove some results about multiply fractured media, and investigate the relationships between various models of fracturing. .

REFERENCES

Backus, G.E. , 1962, Long-wave anisotropy produced by horizontal layering: Journal of Geophysical Reseach **66**, p. 4427-4440.

Muir, F. and Dellinger J. , 1988, A calculus for layered elastic media: SEP–**57**, p. 179-189.

Nichols, D. , 1988, Building ansotropic models : SEP–**57**, p. 191-205

Schoenberg, M. and Muir, F., 1988, A calculus for finely layered anisotropic media: Submitted to Geophysics.

Wolfram, S. , 1988, *Mathematica*^{TM}, A system for doing mathematics by computer: Addison-Wesley

APPENDIX I : Schoenberg/Muir calculus for layered media

The method used is that outlined by Muir in SEP–56 and more fully described by Schoenberg and Muir (1988) and Muir and Dellinger (1988). The calculation of the properties of layered media is related to the work of Backus (1962) and gives the equivalent elastic properties of a layered medium for layering much finer than the seismic wavelength.

Hooke's law for a general elastic medium can be written as,

$$\begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \sigma_4 \\ \sigma_5 \\ \sigma_6 \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} & c_{15} & c_{16} \\ c_{21} & c_{22} & c_{23} & c_{24} & c_{25} & c_{26} \\ c_{31} & c_{32} & c_{33} & c_{34} & c_{35} & c_{36} \\ c_{41} & c_{42} & c_{43} & c_{44} & c_{45} & c_{46} \\ c_{51} & c_{52} & c_{53} & c_{54} & c_{55} & c_{56} \\ c_{61} & c_{62} & c_{63} & c_{64} & c_{65} & c_{66} \end{bmatrix} \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \epsilon_4 \\ \epsilon_5 \\ \epsilon_6 \end{bmatrix}$$

where

$$\begin{bmatrix}\sigma_1\\ \sigma_2\\ \sigma_3\\ \sigma_4\\ \sigma_5\\ \sigma_6\end{bmatrix} = \begin{bmatrix}\sigma_{11}\\ \sigma_{22}\\ \sigma_{33}\\ \sigma_{23}\\ \sigma_{31}\\ \sigma_{12}\end{bmatrix} \quad \textit{and} \quad \begin{bmatrix}\epsilon_1\\ \epsilon_2\\ \epsilon_3\\ \epsilon_4\\ \epsilon_5\\ \epsilon_6\end{bmatrix} = \begin{bmatrix}\epsilon_{11}\\ \epsilon_{22}\\ \epsilon_{33}\\ 2\epsilon_{23}\\ 2\epsilon_{31}\\ 2\epsilon_{12}\end{bmatrix} .$$

are the stress and strain tensors written in the abbreviated subscript notation.

Schoenberg and Muir (1988) and Muir and Dellinger (1988) show that if the following 3×3 matrices are defined for the i^{th} component of a layered medium,

$$\mathbf{C}_{TT_i} = \begin{bmatrix} c_{11_i} & c_{12_i} & c_{16_i} \\ c_{12_i} & c_{22_i} & c_{26_i} \\ c_{16_i} & c_{26_i} & c_{66_i} \end{bmatrix} , \quad \mathbf{C}_{NN_i} = \begin{bmatrix} c_{33_i} & c_{34_i} & c_{35_i} \\ c_{34_i} & c_{44_i} & c_{45_i} \\ c_{35_i} & c_{45_i} & c_{55_i} \end{bmatrix} , \quad \mathbf{C}_{NT_i} = \begin{bmatrix} c_{13_i} & c_{14_i} & c_{15_i} \\ c_{23_i} & c_{24_i} & c_{25_i} \\ c_{36_i} & c_{46_i} & c_{56_i} \end{bmatrix} ,$$

the total thickness of all layers of the i^{th} constituent in the composite is H_i and its density is ρ_i, we can define the following transformation to 'group' coordinates,

$\mathbf{G}_i = [\, g_i(1), g_i(2), \mathbf{g}_i(3), \mathbf{g}_i(4), \mathbf{g}_i(5) \,]$ of two scalars and three 3×3 matrices.

The mapping is given by

$$\begin{bmatrix} H_i \\ H_i\,\rho_i \\ H_i\,\mathbf{C}_{NN_i}^{-1} \\ H_i\,\mathbf{C}_{NT_i}\,\mathbf{C}_{NN_i}^{-1} \\ H_i\,[\,\mathbf{C}_{TT_i} - \mathbf{C}_{NT_i}\mathbf{C}_{NN_i}^{-1}\mathbf{C}_{NT_i}^{T}\,] \end{bmatrix} \rightarrow \begin{bmatrix} g_i(1) \\ g_i(2) \\ \mathbf{g}_i(3) \\ \mathbf{g}_i(4) \\ \mathbf{g}_i(5) \end{bmatrix} ,$$

For any $\mathbf{G}_i$ such that $\mathbf{g}_i(1) \neq 0$ and $\mathbf{g}_i(3)$ is invertible, we can return to the set of physical model parameters by the 'inverse group mapping' from group element $\mathbf{G}_i$ to physical model. This inverse mapping is given by

$$\begin{bmatrix} g_i(1) \\ g_i(2)/g_i(1) \\ g_i(1)\mathbf{g}_i(3)^{-1} \\ \mathbf{g}_i(4)\mathbf{g}_i(3)^{-1} \\ [\mathbf{g}_i(5) + \mathbf{g}_i(4)\mathbf{g}_i(3)^{-1}\mathbf{g}_i(4)^{T}]/g_i(1) \end{bmatrix} \rightarrow \begin{bmatrix} H_i \\ \rho_i \\ \mathbf{C}_{NN_i} \\ \mathbf{C}_{NT_i} \\ \mathbf{C}_{TT_i} \end{bmatrix} .$$

The group has been chosen in this way so that adding the respective scalars and matrices for all constituents of a layered composite and performing the inverse group transformation will give a set of elastic constants that is the long wavelength

equivalent elastic medium.

The method of calculating the properties of fractured media is that described by Schoenberg and Douma (1988) which is shown by Schoenberg and Muir (1988) to correspond to increasing the third group element. A set of fractures is defined by a symmetric 3×3 matrix that is added to the third group element of the rock to be fractured.

APPENDIX II : Schoenberg/Muir notebook

MAIN DEFINITIONS:

```
layer = MakeLayer [ thickness, density, elasticMatrix ]
c =  a + b
a =  c - a
c = SMCrack [ a, b, excess ]
b = Rotate{XYZ}[ a, angle ]
IsTI[ a ]
IsOrth[ a ]
Show [ a ]

Utilities:
elastic matrix = GenTI[x]
elastic matrix = GenOrth[x]
elasticMatrix = ImpedanceMatrix [ density, P_velocity, SV_velocity, SVh_velocity ]
elasticMatrix = LameMatrix [ lambda, mu1, mu2 ]

To load:
<<Schoenberg/Muir.m
```

■

Layer: 5-element list: z, rho, Cnn, Cnt, Ctt.

MakeLayer []: generator
LtoG[] : map layer to group representation
Rotate{XYZ}[] : rotate the elastic constants for a layer
+ : combine layers according to group theory
- : remove layer according to group theory
SMCrack []: crack a layer using Schoenberg/Muir method

IsTI[] : test if transverse isotropic
IsOrth[] : test if orthorhombic
ElasticMatrix[]: get the elastic matrix for a layer
Show[]: formatted printout of layer

```
(* make layer list *)
MakeLayer [ z_, rho_, c_ ] := Layer[ {
     z,
     rho,
     NMatrix[ c ],
     PMatrix[ c ],
     MMatrix[ c ]
     }    ]

(* map layer to group representation *)
Layer/: LtoG [ Layer[layer_] ] := Group[ {
     layer[[1]],
     layer[[1]] layer[[2]],
     layer[[1]] Inverse [layer[[3]]],
     layer[[1]] layer[[4]] . Inverse [layer[[3]]],
     layer[[1]] (layer[[5]] - layer[[4]] . Inverse [layer[[3]]
          Transpose [ layer[[4]] ])
     } ]

(* See if a layer is Transverse Isotropic *)
Layer/: IsTI [ Layer[layer_] ] :=
     IsTI[
          ElasticMatrix[ Layer[layer] ]
     ]

(* See if a layer is orthorhombic *)
Layer/: IsOrth [ Layer[layer_] ] :=
     IsOrth [
          ElasticMatrix[ Layer[layer] ]
     ]

(* rotate the elastic constants for a layer *)

Layer/: RotateX[ Layer[x_], angle_ ] :=
MakeLayer[  x[[1]],
            x[[2]],
            RotateX[
                 ElasticMatrix [ Layer[x] ],
                 angle
            ]
         ]
```

```
Layer/: RotateY[ Layer[x_], angle_ ] :=
MakeLayer[  x[[1]],
            x[[2]],
            RotateY[
                  ElasticMatrix [ Layer[x] ],
                  angle
            ]
         ]

Layer/: RotateZ[ Layer[x_], angle_ ] :=
MakeLayer[  x[[1]],
            x[[2]],
            RotateZ[
                  ElasticMatrix [ Layer[x] ],
                  angle
            ]
         ]

(* format layer list *)
Layer/: Show [ Layer[x_] ] :=
     TableForm [ {
          { "Layer" },
          {"z=", x[[1]]},
          {"rho=", x[[2]]},
          "c=",
          MatrixForm [
               ElasticMatrix [ Layer[x] ]
               ]
          } ]


(* extract the elastic matrix *)
Layer/: ElasticMatrix[ Layer[ a_] ]:=
     FullMatrix [
                  a[[3]],a[[4]],a[[5]]
                  ]

(* combine layer *)
Layer/: Plus [ Layer[a_], Layer[b_] ] :=
     GtoL [
          LtoG [Layer[a]] + LtoG [Layer[b]]
     ]

(* uncombine layer *)
Layer/: Subtract [ Layer[a_], Layer[b_] ] :=
     GtoL [
          LtoG [Layer[a]] - LtoG [Layer[b]]
     ]
```

```
(* crack layer in Schenberg/Muir sense *)
Layer/: SMCrack [ Layer[a_], Layer[b_] , excess_ ] :=
     GtoL[
          SMCrack[
               LtoG[ Layer[a] ],
               LtoG[ Layer[b] ],
               excess
          ]
     ]
```

■

Group: 5-element list: g1,g2,g3,g4,g5 of anisotropic Abelian group

GtoL []: convert group to layer
+ combination operator for Abelian group
- removal operator for Abelian group(* make layer list *)
Show []: formatted printout of group.
SMCrack []: crack a group using Schoenberg/Muir method

```
(* map from group representation to layer*)
Group/: GtoL [ Group[g_] ] := Layer[  {
     g[[1]],
     g[[2]] / g[[1]],
     g[[1]] Inverse [g[[3]]],
     g[[4]] . Inverse [g[[3]]],
     (g[[5]] + g[[4]] . Inverse [g[[3]]] . Transpose [g[[4]]])
          / g[[1]]
     }   ]

Group/: Plus[ Group[a_],Group[b_] ] := Group[a + b]
Group/: Minus[ Group[a_] ] := Group[-a ]
Group/: Subtract[ Group[a_] ,Group[b_]] := Group[a-b ]

(* format group list *)
Group/: Show [ Group[x_] ] :=
     TableForm [ {
          { "Group" },
          {"g(1)=", x[[1]]},
          {"g(2)=", x[[2]]},
          "g(3)=",
          MatrixForm [
               x [[3]]
```

```
            ],
          "g(4)=",
          MatrixForm [
               x [[4]]
               ],
         "g(5)=",
          MatrixForm [
              x [[5]]
              ]
          } ]

Group/: SMCrack[ Group[a_], Group[b_], excess_ ] :=
          Group[{ a[[1]], a[[2]] ,
                  a[[3]] + b[[3]] * excess ,
                  a[[4]],a[[5]] }]
```

■ ElasticMatrix = 6 by 6 array. { Object type: Elmat }

LameMatrix []: generator
ImpedanceMatrix []: generator
GenTI[x] : generate a generic TI matrix using the symbol x.
GenOrth[x] : generate a generic orthorhombic matrix using the symbol x.
Transform[] : apply coordinate transform
IsTI[] : test if transverse isotropic
IsOrth[] : test if orthorhombic
Rotate{XYZ}[]: rotate the coordinate frame
TnsrForm[] : convert to a rank-4 tensor

Submatrices: M, N, P 3 x 3 arrays from ElasticMatrix.
MMatrix []: generator
NMatrix []: generator
PMatrix []: generator
FullMatrix []: regenerate elastic matrix from submatrices

■

low level utility functions

■

error messages