The *cm*-package
A MapleV program for continuum mechanics

Mikael Eriksson and Lars H. Söderholm

September 1999

Abstract

The *cm*-package is a system for calculating tensor objects related to continuum mechanics and for their manipulations. On constructing the package, emphasis has been on creating a simple and easy to understand user's interface. The user who is familiar with continuum mechanics should quickly be able to apply and understand, as well as further expand and customize the package.

A Master Thesis Project at the Department of Mechanics,
Royal Institute of Technology, Stockholm, Sweden, by Mikael Eriksson
Supervisor: Lars H. Söderholm, Associate Professor

Contents

1 Introduction

In section 1.1 a brief review of how and why the *cm*-system was developed is given, while section 1.2 takes up som basic concepts in continuum mechanics and gives an overview of the rest of this user's guide.

1.1 Background

To formulate continuum mechanics in general, one need two spaces, the present configuration (or simply space) and the reference configuration of the body. The basic object in continuum mechanics is the mapping between these two spaces, see [4].

In General Relativity there exists the impressive tool GR-tensor, see Musgrave et al [1]. GRtensor is a MapleV package for the manipulation and calculation of tensor fields in general relativity. For GR-Tensor there is an application called Elasticity [2], which makes it possible to do calculations in nonlinear elasticity. Elasticity is a library to GRtensor.

General Relativity, however, has one space-time only. The idea behind Elasticity is to formally regard the coordinates in the present configuration and the coordinates in the reference configuration as two coordinate systems for the same space. See, e.g., [4]. The mapping from (the coordinates in) the reference configuration to (the coordinates in) the present configuration is thus formally regarded as a coordinate transformation. This way it is possible to calculate many quantities, in particular to set up the equations of equilibrium in the present configuration, see [2].

In reality, however, the present configuration (or simply space) and the reference configuration are two different spaces, see [4] [5]. The reason for this is simply that there are two different distances associated to two neighbouring points (or particles as the nomenclature is in continuum mechanics), the distance in the present configuration and the distance in the reference configuration. To be concrete, in space there is the ordinary Euclidean distance but also the distance of the reference configuration carried over and expressed by the inverse of the left (referential) Cauchy-Green (metric) tensor. Similarly, in the reference configuration, there is the ordinary Euclidean distance, but also the distance of the present configuration carried over, and expressed by the right (referential) Cauchy-Green (metric) tensor.

If one considers the mapping between the two configurations as coordinate transformation, one has two different metric tensor, and thus two ways to raise and lower indices. This becomes particularly troublesome for so-called two-point objects or double tensors, see [6] , living with one foot in each of the spaces.

To formulate the equations of equilibrium in the reference configuration one needs the first Piola-Kirchhoff stress tensor rather than the ordinary Cauchy stress tensor. The first Piola-Kirchhoff stress tensor is an important

example of a two-point object, a tensor mapping between the configurations. Another object of this kind is the deformation gradient. To be able to handle such tensor field, a setting with two different spaces with a mapping between them is the natural one.

The present package is based on this concept of mapping between two different spaces. It can handle tensor field in the present configuration and the reference configuration as well as two-point objects. Much of it has been inspired by the GR-Tensor.
indent This is where the *cm*-system comes in. Being developed with continuum mechanins in mind, the system handles the two different metrics and allows the user to decide in which configuration to perform calculations. furthermore, objects can be placed in either the present or the reference configuration, or they can be mixed and thus placed in both.

1.2 Deformations

On studying deformations, the fundamental relation is the mapping

$$x^k = \chi^k(X^L), \tag{1}$$

connecting the points with coordinates X^L of the body in its reference configuration, with the points with coordinates x^k in the so called present configuration [4] [3] [5].

Throughout this user's guide, we will follow an example of a rectangular block being deformed. The deformation is described by the three equations

$$r = \sqrt{2AX}, \tag{2}$$

$$\phi = BY, \tag{3}$$

$$z = \frac{Z}{AB} - BCY. \tag{4}$$

Here r, ϕ, z are cylindrical coordinates in the present configuration and X, Y, Z cartesian coordinates in the reference configuration. This is one of the universal solutions for incompressible isotropic nonlinearly elastic solids. See Truesdell [4].

Starting with basics, how to install and start up the system is described in section 2. In section 3, how to define the environment is described (i.e. how to choose coordinates and how to define a mapping). In section 4 we look at how objects are being calculated and displayed, while section 5 describes how to perform derivation. The following sections deals with how to expand the system, in section 6 by defining new tensor objects, and in section 8 by explaining how to write your own code in *cm*. The last section 9 is an overview of all procedures and predefined tensor objects that are given with the *cm*-system.

2 Getting started

The *cm*-package is developed for MapleV. How to install the package is described in 2.1 and how to start up the system is described in 2.2

2.1 Installing the *cm*-package

Before installing and starting up the *cm*-system you need to install MapleV. That this has been done will now be assumed.

2.1.1 Installing cm on unix under afs

The stepps of installing the *cm*-system on unix are:

1. Create a directory in which to place *cm*

   ```
   > mkdir cm
   ```

2. Go down in your new directory and download the compressed file to that directory

3. Unpack the compressed file with the commands

   ```
   > gzip -d cm.tar.gz
   > tar xvf cm.tar
   ```

4. Open your *.mapleinit*-file and add a MapleV library path, to the directory containing cm, by including the line

   ```
   libname := ``/afs/.../cm'' , libname:
   ```

   where the ... should be replaced by the path to your home directory.

5. When you unpacked the compressed file, a file *cmmetricpath* was created. Open that file and fill in the missing parts of the path given there.

The installation is now completed.

2.2 Starting up the *cm*-package

A new session with the *cm*-package begins in a MapleV-worksheet with the command:

```
> restart;

> readlib(cmlib);
```

```
> cmstartup();
```

Welcome to the cm-package!
This system was developed by,
Mikael Eriksson
in 1999,
as a Master thesis project.
Supervisor: Lars H. Söderholm
KTH
Department of Mechanics
The system cm is now ready

3 Defining the environment

The *cm*-system can be used strictly as a tensor program, but the main task for which the system was designed, is to use *cm* to calculate tensors in continuum mecanics. This ability to use *cm* for different tasks, means that depending on what calculations you want to perform, defining the environment will have different meanings.

If you just want to use the tensor machinery, defining the environment need only mean to specify the metric for to the system to work in, how this is done is described in section 3.1.2. If instead you want to use *cm* for calculations in continuum mechanics, then defining the environment requires you to specify metrics both for the present and the reference configuration (see sections 3.1.1 and 3.1.2) and to load a mapping or an inverse mapping (see section 3.2), thus connecting the points in the present configuration with the points in the reference configuration.

Following standard continuum mechanics notation, metrics and variables in the reference configuration begin with an upper case letter, while metrics and variables in the present configuration will begin with lower case letters. In particular, a spherical metric in the present configuration will thus be labelled as *spherical*, with independent variables r, ϕ, θ, while a cylindrical metric in the reference configuration will be labelled *Cylindrical*, with independent variables R, Φ, Z.

3.1 Metrics

With the *cm*-package comes a standard set of predefined metric tensors, which are all easy to access and ready to use. If however you wish to use a different metric, other than those predefined, you can easily define new metrics with the system. How that is done is decribed in section 6.2. This section shows how to specify metrics for the system to use.

Specifying the metrics to the system is done with the *cmsetmetric* pro-

cedure. The procedure is interactive and does not take any arguments. We see how this is done both in the present and the reference configuration.

3.1.1 Setting metric in the reference configuration

According to our example with the rectangular block being deformed, we wish to use cartesian coordinates in the reference configuration and the commands to give are:

```
> cmsetmetric();
```

For which configuration do you wish to enter a metric?
Enter:

1, For the reference configuration
2, For the present configuration

```
Enter 1 or 2   <- : 1;
```

Enter metric for the reference configuration:
To see metrics available enter: avail

```
metric name      <- : Cartesian;
```

Metric in the reference configuration is now:
Cartesian
$[X, Y, Z]$

We have now set metric in the reference configuration and proceed by entering metric for the present configuration.

3.1.2 Setting metric in the present configuration

Our choice for the metric in the present configuration is cylindrical and we give the commands:

```
> cmsetmetric();
```

For which configuration do you wish to enter a metric?
Enter:

1, For the reference configuration
2, For the present configuration

```
Enter 1 or 2   <- : 2;
```

Enter metric for the present configuration:
To see metrics available enter: avail

```
metric name      <- : cylindrical;
```

Metric in the present configuration is now:
cylindrical
$[r, \phi, z]$

If you don't know what metrics are available to the system, on entering the command *avail* instead of a metric name, the available metrics will be displayed. For instance, if we instead of entering *cylindrical*, had entered *avail*, the response would have been:

```
metric name      <- : avail;
```

{*cylindrical, spherical, polar, cartesian*}

indicating that, for the present configuration, those four metrics are known to *cm*. Thus letting you know that by entering *cylindrical*, the preferred metric will be set.

3.2 Mappings

To know how the coordinates in the reference configuration and the present configuration are related to each other, the *cm*-system must be given a mapping (defining coordinates in the present configuration in terms of the coordinates in the reference configuration) or an inverse mapping (going the other way around). This is done with the *cmloadmapping* or the *cmloadinvmapping* procedures respectively. Whether you should load a mapping or an inverse mapping depends on which coordinates you wish to use as your independent variables. If you want to work in the reference configuration, using the reference coordinates as independent variables, then a mapping should be loaded. If instead you wish to perform your calculations in the present configuration, an inverse mapping should be loaded.

3.2.1 Loading a mapping with the *cmloadmapping* procedure

We wish to enter the mapping described by the equations (2)-(4). The *cmloadmapping* procedure takes as argument a list of the mapping to be defined. In our example, the command to give is therefore:

```
> cmloadmapping([r=sqrt(2*A*X),phi=B*Y,z=Z/(A*B)-B*C*Y]);
```

You are now working in the reference configuration
Your independent variables are:
$[X, Y, Z]$

Note that after performing this command, the *cm*-system will use *X,Y,Z* as independent variables.

3.2.2 Loading an inverse mapping with the *cmloadinvmapping* procedure

If we instead would like to work with the independent variables r, ϕ, z, then an inverse mapping should be loaded. That means that the equations (2)-(4) first must be expressed in terms of r, ϕ, z. This is done by

$$X = \frac{r^2}{2A}, \tag{5}$$

$$Y = \frac{\phi}{B}, \tag{6}$$

$$Z = ABz + ABC\phi. \tag{7}$$

Like the *cmloadmapping* procedure, the argument must be a list containing the inverse mapping to be defined and the command to give is therefore:

```
> cmloadinvmapping([X=r^2/(2*A),Y=phi/B,Z=A*B*z+A*B*C*phi]);
```

You have allready loaded a mapping. This operation is ignored!

The *cm*-system recognizes that we have already loaded a mapping, and to prevent errors from being made, we are not allowed to additionaly load an inverse mapping, or to load a second mapping for that matter. In order to load a new mapping we must first perform a restart as described in section 2.2.

4 Calculating and displaying tensor objects

With the *cm*-package comes the definitions for fundamental tensor objects in continuum mechanics, such as the deformation gradient **F**, the right (referential) Cauch-Green deformation tensor **C** and the left (present) Cauchy-Green deformation tensor **B** among others. The tensors can be displayed in terms of their components in a coordinate basis. For orthogonal coordinates they can also be displayed through their physical components. A summary of all predefined objects are found in section 9.4, while how to calculate and display them is described here.

Displaying the different objects is done with the *cmdisplay* procedure, which uses the *cmCalc* procedure to calculate the objects (i.e. you don't need to actually calculate the objects before displaying them, the *cm*-system takes care of that for you. How *cmCalc* works is described in section 8.2.1).

4.1 *cmdisplay*

In continuum mechanics, in order to raise/lower indices and perform derivations correctly, keeping track of in which configuration an object is placed is of great importance. The notation in the *cm*-system is therefore built to help the user to separate indices in the two configurations and prevent errors from being done. To illustrate how this all works, some examples are illustrative.

4.1.1 Tensor components in coordinate basis

Indices in the present configuration are labelled *up*, meaning a contravariant index, and *dn* meaning a covariant index. In the reference configuration, contravariant and covariant indices are labelled *Up* and *Dn* respectively.

Example Given a mapping as described by equation (1), the deformation gradient **F** is defined by

$$F^{k}_{\ L} = x^{k}_{\ ,L}. \tag{8}$$

We see that the first index of **F** is contravariant and placed in the present configuration, while the second index is covariant in the reference configuration. In section 3.2.1 we loaded a mapping corresponding to the equations (2)-(4). The command to give to display the components of **F** corresponding to that mapping will be:

```
> cmdisplay(F(up,Dn));
```

$$F^{r}_{\ X} = \frac{1}{2}\frac{\sqrt{2}A}{\sqrt{AX}}$$

$$F^{\phi}_{\ Y} = B$$

$$F^{z}_{\ Y} = -BC$$

$$F^{z}_{\ Z} = \frac{1}{AB}$$

Note that *cm* only shows the nonvanishing components.

Example The right (referential) Cauchy-Green deformation tensor **C** is defined by

$$C_{KL} = g_{kl} F^{k}{}_{K} F^{l}{}_{L}, \tag{9}$$

where g_{kl} are the components of the metric tensor in the present configuration. With both indices in the reference configuration, to display the contravariant components of **C** we give the command:

```
> cmdisplay(C(Up,Up));
```

$$C^{XX} = \frac{1}{2}\frac{A}{X}$$

$$C^{YY} = 2AXB^2 + B^2C^2$$

$$C^{YZ} = -\frac{C}{A}$$

$$C^{ZY} = -\frac{C}{A}$$

$$C^{ZZ} = \frac{1}{A^2B^2}$$

Note that though **C** is defined with covariant indices, we could immediately display it's contravariant components. In fact we can raise and lower indices as we like; the *cm*-system will take care of the calculations for us.

Example The left (spatial) Cauchy-Green deformation tensor **B** is defined by

$$B^{kl} = G^{KL} F^{k}{}_{K} F^{l}{}_{L}. \tag{10}$$

By now you should be able to give the command to display the components of **B** . Instead let us now see what happens when giving an erroneous command.

```
> cmdisplay(B(Up,up));
```

The object could not be calculated

This is not an existing object since both indices should be in the present configuration, and therefore nothing is calculated.

4.1.2 Physical components of a tensor

Given a general second order tensor in the present configuration, in an orthogonal basis, we define the physical components of the tensor by

$$A_{<i><j>} = \sqrt{g_{ii}}\sqrt{g_{jj}}A^{ij} = \sqrt{g^{ii}}\sqrt{g^{jj}}A_{ij} \qquad \text{(no sum over i, j)}$$

Similary one can define the physical components for tensors in the reference configuration. In *cm*, physical components are labelled *ph* and *Ph* for indices in the present and reference configuration respectively.

Example Let's look at the physical components of **B**.

```
> cmdisplay(B(ph,ph));
```

$$B_{<r><r>} = \frac{1}{2}\frac{A}{X}$$

$$B_{<\phi><\phi>} = 2AXB^2$$

- - - - - -

We have here chosen not to display all components of **B**.

5 Derivation

Calculating derivatives of tensor objects is quite easy. Both partial and covariant derivations can be performed in an intuitive way.

5.1 Partial derivation

Partial derivation with respect to the present variables are done with, *pdn*, while partial derivation with respect to the reference variables is done with *Pdn*.

Example Let's look at the partial derivative of the metric tensor in the present configuration **g**, i.e. let's calculate

$$g_{kl,m} = \frac{\partial g_{kl}}{\partial x^m}.$$

```
> cmdisplay(g(dn,dn,pdn));
```

$$g_{\phi\phi,r} = 2\sqrt{2} * \sqrt{AX}$$

At first this result may seem a bit confusing. Why should the components of the metric tensor **g** be expressed in the reference variable X? When loading our mapping in 3.2.1, we chose to work in the reference configuration, i.e. $g_{kl} = g_{kl}(X^K)$ and what has really been calculated is

$$g_{kl,m} = g_{kl,K}(F^{-1})^K{}_m = g_{kl,K}\frac{\partial X^K}{\partial x^m}$$

That this indeed is the case can be verified by displaying the components of **g** and of the object $\mathbf{F^{-1}}$, with the commands g(dn,dn) and Finv(Up,dn) respectively.

5.2 Covariant derivation

When calculating the covariant derivative of a tensor, the affinities are needed. The affinities in the present configuration are defined by

$$\Gamma_{klm} = \frac{1}{2}(g_{kl,m} + g_{km,l} - g_{lm,k}) \tag{11}$$

Here g_{kl} is the components of the metric tensor in the present configuration. In the reference configuration, the affinities are defined by:

$$\widetilde{\Gamma}_{KLM} = \frac{1}{2}(G_{KL,M} + G_{KM,L} - G_{LM,K}) \tag{12}$$

and the metric tensor in the reference configuration is seen to be labelled **G**.

Example Let us, before proceeding with our covariant derivation, take a look at the affinities in both the present and the reference configuration. The affinities are also denoted Christoffel symbols and in *cm* they are therefore labelled *chr* and *Chr* for present and reference configuration respectively.

```
> cmdisplay(chr(up,dn,dn));
```

$$\Gamma^r{}_{\phi\phi} = -\sqrt{2}\sqrt{AX}$$

$$\Gamma^\phi{}_{r\phi} = \frac{1}{2}\frac{\sqrt{2}\sqrt{AX}}{AX}$$

$$\Gamma^\phi{}_{\phi r} = \frac{1}{2}\frac{\sqrt{2}\sqrt{AX}}{AX}$$

```
> cmdisplay(Chr(Up,Dn,Dn));
```

All components are zero

Here we notice that in the reference configuration, the affinities are denoted with an initial upper case letter, *Chr*, while in the present configuration, they are just labelled *chr*. We also notice that the affinities in the reference configuration are all zero, which shouldn't come as a surprise, since we are working with cartesian coordinates there.

Example In section 4.1 we calculated the components of **C**, and we can now look at its covariant derivatives. The command to give is:

```
> cmdisplay(C(Dn,Dn,Cdn))
```

$$C_{XX;X} = -\frac{1}{2}\frac{A}{X^2}$$

$$C_{YY;X} = 2AB^2$$

On a closer look at this result, we recognize this as just being the partial derivative of C, as it should be, since all the affinities in the reference configuration are zero, thus making no contribution to the covariant derivation of C.

Example If we now look at the covariant derivative of **B**, the affinities in the present configuration will make contributions.

```
> cmdisplay(B(up,up,cdn))
```

$$B^{rr}{}_{;r} = -\frac{1}{2}\frac{\sqrt{2}\sqrt{AX}}{X^2}$$

$$B^{r\phi}{}_{;\phi} = -AXB^2 + \frac{1}{8}\frac{\sqrt{2}\sqrt{\frac{1}{AX}}}{X}$$

- - - - - -

and we have once again chosen not to display all components.

Example Covariant differentiation can also be performed on mixed objects, e.g. **F**.

```
> cmdisplay(F(up,Up,Cdn))
```

$$F^{rX}{}_{;X} = -\frac{1}{4}\frac{\sqrt{2}A^2}{(AX)^{3/2}}$$

$$F^{rY}{}_{;Y} = -\sqrt{2}\sqrt{AX}B^2$$

- - - - - -

6 Expanding the *cm*-system

We will now leave our example of deformation of the rectangular block and, through a number of examples, show how the *cm*-system easily can be expanded by adding new tensor objects as well as new metrics. New tensors can be defined in terms of previously existing tensors as well as by entering the components of the tensor to be defined. More about how this is done is described in section 6.1. In section 6.2, how to define and add new metrics to the *cm*-system is decribed.

6.1 Defining new tensors with the *cmdefine* procedure

Defining new tensor objects can be done in two ways, both by entering the components of the object to be defined, but also by expressing new tensors in terms of previously existing tensors.

6.1.1 Defining new tensors by entering their components

As an example, let us define a vector field **u** in space, the contravariant components of which are given by

$$u^r = r^2 * sinh\phi, \tag{13}$$

$$u^\phi = cos\phi. \tag{14}$$

We could for example be interested in calculating its covariant derivatives, and wish to leave the calculations to *cm*. First we must set metric in the present configuration to be *polar*. The commands to give were described in section 3.1, and we will now assume that this has been done. To define a vector field **u** through it contravariant components, the command to give is:

```
> cmdefine(`u(^k)`);
```

Enter the components of the tensor
u
The components should be entered as a list

```
list of components <- : [r^2*sinh(phi),cos(phi)];
```

The object:
u(up)
has now been defined!

On defining an object, the argument to *cmdefine* must be of the type Maple-string, thus the: `s surrounding the object. Further, we see that a ^ in front of a lower case *k* is interpreted as a contravariant index in the present configuration. The tensor *u*, is now defined and can be handled as all predefined objects, the system makes no difference between them.

6.1.2 Defining new tensors in terms of existing tensors

The *cm*-system allows you to define new objects in terms of previously existing objects by using a simple tensor notation. This is demonstrated through a number of examples.

Example We would like to define a tensor **A** as the square of **B**. The command to give is then.

```
> cmdefine(‘A(k l):=B(k ^m)*B(m l)‘);
```

The object:
A(dn, dn)
has now been defined!

Allowed indices to use for the present configuration are the letters: (k, l, m, n, o, p, q, r, s, t, u, v) and a ^ in front of an index means that that is a contravariant index. Once the object *A(dn,dn)* is defined, it is treated like all predefined objects, and you display its components and perform derivations on it, exactly as has been demonstrated in sections 4 and 5.

Example Two equal indices means contraction over those indices. We demonstrate this by creating our own definition for the divergence of **B**.

```
> cmdefine(‘divB(^k):=B(^k ^m;m‘);
```

The object:
divB(up)
has now been defined!

The semicolon means covariant derivation and since the second and third indices are both labelled m, after performing the derivation on **B**, a contraction over the second and third indices is done.

Example Let us as a last example define **B** on our own. As the object name **B** is already taken, we call this object **Bnew**. The definition is given by equation (10). The command to give is.

```
cmdefine(‘Bnew(^k ^l):=G(^L ^K)*F(^k K)*F(^l L)‘);
```

The object:
Bnew(up,up)
has now been defined!

The result pretty much talks for itself.

6.2 Defining new metrics with the *cmmakemetric* procedure

Defining new metrics in the *cm*-system is done with the *cmmakemetric* procedure. The procedure is interactive and takes no argument. On using it you will see that there are two different ways to go about. If you already know the components of your new metric, you define it by just entering those components. The other way is by entering a transformation to or from the variables of an already known metric.

6.2.1 Defining new metrics by entering their components

We wish to define a twodimensional cartesian metric, but with variables labelled $X1, X2$, instead of X, Y. This is done with the commands.

```
> cmmakemetric();
```

Do you wish to define the new metric by:

1, Entering the covariant components of the metric tensor, or
2, By entering a transformation to/from a known metric?

```
Enter 1 or 2 <- : 1;
```

Enter the name of the metric to be defined:

```
metric name  <- : CartesianX1X2;
```

Enter configuration (i.e ref / pres):

```
configuration  <- : ref;
```

Enter a list of the variables (e.g [x,y,z]):

```
variables      <- : [X1,X2];
```

Enter the components of the metric tensor G:
Enter component:
$G[X1]\ [X1]$

```
component      <- : 1;
```

Enter component: $G[X1]\ [X2]$

```
component      <- : 0;
```

Enter component: $G[X2]\ [X2]$

```
component      <- : 1;
```

Do you wish to save the metric:
CartesianX1X2
and make it a predefined metric (yes/no):

```
save metric ?  <- : no;
```

For the reference configuration:
The metric
CartesianX1X2
has now been defined

Since the metric tensor is symetric, we don't need to enter all components. The system will tell you what components to enter, so be sure to enter the component actually requested. Since we did not choose to save this metric, once we terminate our session in *cm*, by performing a MapleV *restart*, the system will have forgotten all about it.

If you now wish to use this metric in your current session you need to set it as the present metric with the *cmsetmetrics* procedure as described in section 3.1

In the next section we will demonstrate how to load and add a metric to the library of metrics in *cm*, and what that means.

6.2.2 Defining new metrics by entering a transformation

We are given an elliptical transformation and want to calculate the components of the elliptical metric. The transformation is described by the equations

$$x = \cosh\mu\cos\phi, \tag{15}$$

$$y = \sinh\mu\sin\phi. \tag{16}$$

The transformation is thus given from our new variables μ and ϕ, to the cartesian variables x and y. The commands to give are:

```
> cmmakemetric();
```

Do you wish to define the new metric by:

1, Entering the covariant components of the metric tensor, or
2, By entering a transformation to/from a known metric?

```
Enter 1 or 2   <- : 2;
```

Enter the name of the metric to be defined:

```
metric name    <- : elliptical2D;
```

Enter configuration (i.e ref / pres):

```
configuration  <- : pres;
```

From/To which metric do you wish to define the transformation:

```
metric name    <- : cartesian2D;
```

Enter the independent variables in your new metric

```
variables      <- : [mu,phi];
```

Enter the transformation as a list

```
transformation <- : [x=cosh(mu)*cos(phi),y=sinh(mu)*sin(phi)];
```

Do you wish to save the metric:
elliptical2D
and make it a predefined metric (yes/no):

```
save metric ?  <- : yes;
```

For the present configuration:
The metric
elliptical2D
has now been defined
and saved for future use

The metric *elliptical2D* has now been defined and saved. That means that the library directory containing the files of the metrics, has been added an extra file containing the definitions for the elliptical2D metric. Exiting *cm* and MapleV will not alter this definition and the next time you start the system, the elliptical2D metric is ready to use.

7 Special *cm*-procedures and operators

7.1 *cmalter*

Maple provides several simplification and alteration routines. The most frequently used routines have been integrated in the *cm*-system, thus making it simple for the user to apply those routines to the various *cm*-objects. The *cmalter* procedure takes as argument a list of the object names you wish to perform alteration routines on. As an example let us look at the twodimensional elliptical metric, that was defined in section 6.2. We give the following command:

```
> restart;
```

```
> cmsetmetric();
```

For which configuration do you wish to enter a metric?
Enter:

1, For the reference configuration
2, For the present configuration

```
Enter 1 or 2   <- : 2;
```

Enter metric for the present configuration:
To see metrics available enter: avail

```
metric name      <- : elliptical2D;
```

Metric in the present configuration is now:
elliptical2D
$[\mu, \theta]$

```
> cmdisplay(g(dn,dn));
```

$$g_{\mu\mu} = \sinh^2\mu\cos^2\theta + \cosh^2\mu\sin^2\theta$$

$$g_{\theta\theta} = \sinh^2\mu\cos^2\theta + \cosh^2\mu\sin^2\theta$$

We immediately sense that there should be a way of expressing these components in a more compact way. What we need to do is to apply some trigometric rules to these expression. We go about it by calling the *cmalter* procedure, giving a list containing g as its argument:

```
> cmalter([g]);
```

Enter a list of numbers corresponding to your preffered
alteration routines
To see which routines are available, Enter: avail

```
> routines    <- :  avail;
```

The following routines are available

1, *For* $simplify$
2, *For* $expand$
3, *For* $factor$
4, *For* $simplify[trig]$
5, *For* $simplify[power]$
6, *For* $simplify[sqrt]$
7, *For* $combine[radical, symbolic]$

```
> routines    <- :  [4];
```

alteration done in the following order:
$[simplify[trig]]$

Let us look at the metric to see if it made any difference:

```
> cmdisplay(g(dn,dn));
```

$$g_{\mu\mu} = -\cos^2\theta + \cosh^2\mu$$

$$g_{\theta\theta} = -\cos^2\theta + \cosh^2\mu$$

Indeed, much nicer.

7.2 *cmsubstitute*

Substitutions of constants, variables or expressions are done with the *cmsubstitute* procedure. As argument it takes two lists. The first list should contain the names of the objects to perform substitutions on, and the second list should contain the expressions defining the substitutions.

Example In section 4.1 we looked at the components of **C**. We could be interested in finding out what effect setting the constant C to zero would have. To see how this works, you should once again define the environment as we did in section 3, i.e. to enter the mapping as defined by the equations (2)-(4). Afterwards we give the command:

```
> cmsubstitute([C],[C=0]);
```

and we see what happened:

```
> cmdisplay(C(Dn,Dn));
```

$$C_{XX} = \frac{1}{2}\frac{A}{X}$$

$$C_{YY} = 2AXB^2$$

$$C_{ZZ} = \frac{1}{A^2B^2}$$

Indeed, the constant C was set to zero.

7.3 *cmgetcomponent*

Extracting information out of the *cm*-system is done with the procedure *cmgetcomponent*. The procedure takes two arguments. Firstly, the object from which to extract the component. Secondly a list containing the indices specifying what component of the object you want to get.

Example We want to get the component ${C_X}^X$ and assign it to the variable CXX. The command to give is:

```
> CXX := cmgetcomponent(C(Dn,Up),[X,X]);
```

$$CXX := \frac{1}{2}\frac{A}{X}$$

The value of ${C_X}^X$ has now been stored in the variable CXX.

7.4 *Operators*

How to call an operator in *cm* is described in section 9.1. Here we give some examples:

7.4.1 Invariants

The invariants of a tensor $\mathbf{A}$ are defined by

$$det(\mathbf{A} - \lambda\mathbf{1}) = -\lambda^3 + I_A\lambda^2 - II_A\lambda + III_A \tag{17}$$

In *cm*, the first tensor invariant I, is calulated with the operator *I1*. Let's continue to explore $\mathbf{C}$ by calculating its first tensor invariant. We give the command:

```
> cmdisplay(I1[C]);
```

$$I_C = \frac{1}{2}\frac{A}{X} + 2AXB^2 + \frac{B^4C^2A^2 + 1}{A^2B^2}$$

and similary the tensor invariants II and III are calculated using the operators *I2* and *I3* respectively.

7.4.2 Divergence

The divergence of a tensor is calculated with the operators *div* and *Div*. The initial d and D here indicates the variables with respect to which the derivation will be performed, the present variables or the reference variables respectively. The divergence of **C** will be calculated using the operator *Div* and the command to give is:

```
> cmdisplay(Div[C](Up));
```

$$(Div\ C)^X = -\frac{1}{2}\frac{A}{X^2}$$

We notice that on calling the operator, we attached, (Up), after the sequence $Div[C]$, this since the divergence of a second order tensor gives a first order tensor. Similary for **B** we have:

```
> cmdisplay(div[B](dn));
```

$$(div\ B)_r = -\frac{1}{2}\frac{\sqrt{2}\sqrt{AX}}{X^2}$$

8 Programming in *cm*

The *cm*-system gives you the ability to define your own objects by constructing MapleV procedures. The procedures should be placed in a special file, which is to be read into the system with the *cmreadlib* procedure, how this is done is described in section 8.1. How to construct a procedure in *cm* is described in 8.2.

8.1 Including your definitions in the *cm*-system

Your code should be placed in a file that is to be read with the *cmreadlib* procedure. Such a file must contain two things. First it must contain the actual procedures with the definitions for how to calculate the objects. Second it must contain a MapleV-table labelled `cm_NewObjDef`, with the definitions for the objects themselves, their names, the procedure being defined, among other things.

This may seem a bit complicated, but it really is quite easy. We want to construct the definitions for two objects and their names are to be $Obj1$ and $Obj2$. $Obj1$ is a second order tensor in the present configuration defined through its covariant indices, while $Obj2$ is a first order tensor in the reference configuration defined through its contravariant indices. We place the code in a file named *newobjects*, and the structure of the file should be:

```
cm_NewObjDef := table([
   Obj1(up,up) = table([
   cm_OD_objtable = cm_Obj1upup_,
   cm_OD_calcfunc = cmCalcObj1,
   cm_OD_objsymbol = Lambda,
   cm_OD_positionlist = [up,up],
   cm_OD_config = pres,
   cm_OD_objvariants = [Obj1(dn,up),Obj1(up,dn),Obj1(up,up)]
   ]),
   Obj2(Dn) = table([
   cm_OD_objtable = cm_Obj2Dn_,
   cm_OD_calcfunc = cmCalcObj2,
   cm_OD_objsymbol = Q,
   cm_OD_positionlist = [Dn],
   cm_OD_config = ref,
   cm_OD_objvariants = [Obj2(Up)]
   ])
]);
cmCalcObj1 := proc()
   ***
end;
cmCalcObj2 := proc()
   ***
end;
```

Once the file *newobjects* has been created, and the procedures *cmCalcObj1* and *cmCalcObj2* have been coded, updating the *cm*-system with those definitions is done with the command:

```
> cmreadlib(newobjects);
```

8.2 Writing your own procedures in *cm*

When making your own code in *cm*, there are some important *cm*-procedures and *cm*-variables you need to use and be able to handle. How they are treated in actual *cm*-code can be seen in section B. Their functions and how to use them are described in this section:

8.2.1 cmCalc

When working with *cm* in a MapleV worksheet, you don't need to think about calculating the different objects before displaying them, the system will take care of that for you, but when programming in *cm*, you must make sure that an object has been calculated before attempting to access its components. This is done with the *cmCalc* procedure. As an example,

in the code for calculating **B** (se section B.2), we need the components of **F**, but before accessing them, they must first be calculated. That is why we include the line:

```
--
cmCalc(F(up,Dn));
--
```

in our code before getting the components.

8.2.2 cmGetObjTable

In *cm*, all information is stored as MapleV-tables. To access the table containing the components we want, we use the procedure *cmGetObjTable*. In our code for **B**, we access the table containing the components of **F** with the line

```
--
Ftable := cmGetObjTable(F(up,Dn));
--
```

and we choose to store the table in the local variable *Ftable*.

8.2.3 cmGetPresVar and cmGetRefVar

The variables used in the present and reference configuration are accessed with the procedures *cmGetPresVar* and *cmGetRefVar* respectively. The procedures will return a list containing the variables. When we in our code for **F** (se section B.1) need the variables in the reference configuration, we access them with the line:

```
--
Diffvar := cmGetRefVar();
--
```

and we assign them to the local variable *Diffvar*, indicating that they will be used for differentiation with respect to.

8.2.4 cmGetMetricDim

The dimension of the metrics being used is accessed with the procedure *cmGetMetricDim*. The procedure takes no argument and will return a positive integer.

8.2.5 cmSetObjTable

Storing information in the system is done with the *cmSetObjTable* procedure. The procedure takes two arguments. The first is the object and the second is a table containing the components of the object. In our code for **B** (se section B.2), the line on which information is stored looks like:

```
--
cmSetObjTable(B(up,up),Btable);
--
```

It is of great importance that the variable being stored is of the type MapleV-table. More about such objects can be found in the MapleV *Help* library.

8.2.6 cmCheckIfNomapping

Until a mapping or an inverse mapping has been loaded, calling the procedure *cmCheckIfNomapping* will return *true*. Ater a mapping or an inverse mapping has been loaded, the procedure will return *false*.

8.2.7 cmCheckIfMapping and cmCheckIfInvmapping

After a mapping has been loaded with the *cmloadmapping* procedure, calling the procedure *cmCheckIfMapping* will return *true*. After an inverse mapping has been loaded with the *cmloadinvmapping* procedure, calling the procedure *cmCheckIfInvmapping* will return *true*. If no mapping or inverse mapping has been loaded, the procedures will return *false*.

Examples of how these procedures can be used are found in the code for **F** and **B** in sections B.1 and B.2.

9 Overview of the *cm*-system

This section is an overview of all the procedures, objects and operators that come with the *cm*-package.

9.1 Syntax:

In the *cm*-system, distinctions are made between tensor objects, their names and tensor operators.

Tensor objects and their names. Tensors are represented by *cm*-objects, and a *cm*-object is the components of a tensor of zeroth or higher order. For example, the deformation gradient **F** is defined through the object F(up,Dn), but all possible variants with indices up and down are also *cm*-objects, referring to the same tensor, i.e. F(dn,Dn), F(dn,Up), F(up,Up) and F(ph,Ph)

are all *cm*-objects. The name of the object is often equivalent to its symbol, thus making F the *cm*-object name of the deformation gradient. To a zeroth order tensor, there is no difference between the object and its name.

Syntax of a zeroth order tensor object:

$$Object = Objectname$$

Syntax of a higher order tensor object:

$$Object = Objectname(Indices)$$

Syntax of indices Indices should be entered with commas between them:

$$Indices = Index1, Index2, ...$$

Indices in the present configuration known to *cm* are:

$$dn, up, ph, pdn, pup, cdn, cup, pph, cph$$

Indices in the reference configuration known to *cm* are:

$$Dn, Up, Ph, Pdn, Pup, Cdn, Cup, Pph, Cph$$

All tensors, their names and through which object they are defined are found in table 9.4.

Tensor operators. With the *cm*-system comes the ability to calculate the invariants and divergence of tensors. This is done with the operators that are listed in 9.3. The *cm*-operators takes as argument within square brackets the name of the object to act on. A typical command will thus have the form, Op[Name], which is evaluated as a *cm*-object name.

Syntax of operational call:

$$Op[Name] \rightarrow Objectname$$

More about operators can be found in section 7.4

Further notations. In the following tables, an upper case O refers to a *cm*-object, an upper case N refers to a *cm*-objectname, a lower case e refers to an equation on the form *variable=expression*.

9.2 Procedures

Procedure	Calling sequence	Described in
cmCalc	cmCalc(O)	8.2.1
cmdisplay	cmdisplay(O)	4.1
cmloadmapping	cmlodmapping([Mapping])	3.2.1
cmloadinvmapping	cmlodinvmapping([Invmapping])	3.2.2
cmsetmetric	cmsetmetric()	3.1
cmmakemetric	cmmakemetric()	6.2
cmalter	cmalter([N1,N2,..])	7.1
cmsubstitute	cmsubstitute([N1,N2,..],[e1,e2,..])	7.2
cmgetcomponent	cmgetcomponent(O,[variables])	7.3

9.3 Operators

Operator	Operator name	Calling sequence
First tensor invariant	I1	I1[N]
Second tensor invariant	I2	I2[N]
Third tensor invariant	I3	I3[N]
Divergence	Div	Div[N]
Divergence	div	div[N]

9.4 Predefined tensor objects

Tensor	Object name	Defined by object
Mapping	*chi*	chi(up)
Inversemapping	*chiinv*	chiinv(Up)
Deformation gradient	*F*	F(up,Dn)
Inverse deformation gradient	*Finv*	Finv(Up,dn)
Right (referential) Cauchy-Green deformation tensor	*C*	C(Dn,Dn)
Inverse right Cauchy-Green deformation tensor	*Cinv*	Cinv(Up,Up)
Left (spatial) Cauchy-Green deformation tensor	*B*	B(up,up)
Inverse left Cauchy-Green deformation tensor	*Binv*	Binv(dn,dn)
Metric in present configuration	*g*	g(dn,dn)
Metric in reference configuration	*G*	G(Dn,Dn)
Affinities in present configuration	*chr*	chr(dn,dn,dn)
Affinities in reference configuration	*Chr*	Chr(Dn,Dn,Dn)
Kronecker delta in present configuraton	*id*	id(up,dn) id(dn,up)
Kronecker delta in reference configuraton	*Id*	Id(Up,Dn) Id(Dn,Up)
Permutation symbol in present configuraton	*prm*	prm(dn,dn,dn) prm(up,up,up)
Permutation symbol in reference configuration	*Prm*	Prm(Dn,Dn,Dn) Prm(Up,Up,Up)
Permutation tensor in present configuration	*eta*	eta(dn,dn,dn) eta(up,up,up)
Permutation tensor in reference configuration	*Eta*	Eta(Dn,Dn,Dn) Eta(Up,Up,Up)

10 Conclusion

The *cm*-package was created in order to provide an easy to use MapleV-program for calculating objects in continuum mechanics. The design of the user‘s interface has been inpired by that of GRtensor and some subroutines uses the MapleV-tensor package for calculations. The main tasks, to be able to handle mixed objects and to perform calcuations in both the present and the reference configuration, have been reached. This is where *cm*, to the best of my knowledge, provides something new.

A Basic concepts in continuum mechanics

A.1 Curvilinear coordinates

Points in space are written $\mathbf{x}$, and are considered functions of the general curvilinear coordinates x^i

$$\mathbf{x} = \mathbf{x}(x^i). \tag{18}$$

Coordinate basis Given a set of curvilinear coordinates x^i we define the basis vectors

$$\mathbf{e}_i = \frac{\partial \mathbf{x}}{\partial x^i}. \tag{19}$$

These basis vectors will in general not be orthogonal or normalized and they are referred to as a *coordinate basis*.

Covariant and contravariant components A vector can be caracterized either by its covariant or its contravariant components. The covariant components of a vector $\mathbf{v}$, denoted v_i, are defined by the scalar product between the vector and a basis vector

$$v_i = \mathbf{e}_i \cdot \mathbf{v}. \tag{20}$$

The contravariant components v^i of that vector, are defined by the relations

$$\mathbf{v} = v^i \mathbf{e}_i. \tag{21}$$

Similary one defines co- and contravariant components for higher order tensors. In general, the co- and contravariant components will not be the same.

The metric tensor We define the covariant components of the metric tensor by

$$g_{ij} = \mathbf{e}_i \cdot \mathbf{e}_j. \tag{22}$$

The contravariant components of the metric tensor are defined by

$$g^{ik} g_{kj} = \delta^i_j. \tag{23}$$

We raice covariant indices with

$$v^i = g^{ij} v_j. \tag{24}$$

We lower contravariant indices with

$$v_i = g_{ij} v^j. \tag{25}$$

A.2 Deformations

Coordinates in the reference configuration are written X^L, while coordinates in the present configuration are written x^k. We define the mapping as

$$x^k = \chi^k(X^L). \tag{26}$$

A.2.1 The deformation gradient

The deformation gradient $\mathbf{F}$ maps a small position vector $\mathrm{d}\mathbf{X}^K$ in the reference configuration, onto a positionvector $\mathrm{d}\mathbf{x}^k$ in the present configuration. We define its components by

$$F^i{}_K = \frac{\partial x^i}{\partial X^K} = x^i{}_{,K}. \tag{27}$$

A.2.2 The Cauchy-Green deformation tensors

We are interested in how a length in the reference configuration has been stretched into a length in the present configuration. To that we need to divide $\mathbf{F}$ into one part containing the rotation of the mapping and one part containing the deformation of the mapping. Using the polar decomposition theorem [5] we write $\mathbf{F}$ as

$$\mathbf{F} = \mathbf{RU} = \mathbf{VR}, \tag{28}$$

where $\mathbf{R}$ is orthogonal and $\mathbf{U}$ and $\mathbf{V}$ are symmetric and positive-definite. Using this decomposition we define two tensors

$$\mathbf{C} = \mathbf{U}^2 = \mathbf{F}^T\mathbf{F}, \tag{29}$$

$$\mathbf{B} = \mathbf{V}^2 = \mathbf{F}\mathbf{F}^T. \tag{30}$$

$\mathbf{C}$ is called the right (referential) Cauchy-Green deformation tensor and $\mathbf{B}$ is called the left (spatial) Cauchy-Green deformation tensor. Their components in a coordinate basis are

$$C_{KL} = g_{kl}F^k{}_K F^l{}_L, \tag{31}$$

$$B^{kl} = G^{KL}F^k{}_K F^l{}_L. \tag{32}$$

A.3 Covariant derivative

When differentiating vector and tensor fields, we must consider that, on moving between points in space, the basis vectors will in general change. This will cause additional terms in the differential, other than the partial derivatives, to contribute.

The gradient of a vector The components of the gradient of a vector field $\mathbf{v}$ in space are

$$v^i{}_{;j} = v^i{}_{,j} + \Gamma^i{}_{kj} v^k. \tag{33}$$

The coefficients $\Gamma^i{}_{kj}$ are called the affinities in the present configuration and are defined through

$$\Gamma^k{}_{lm} = g^{ki}\Gamma_{ilm} = g^{ki}\frac{1}{2}(g_{il,m} + g_{im,l} - g_{lm,i}). \tag{34}$$

Similary we define the affinities in the reference configuration by

$$\widetilde{\Gamma}_{KLM} = \frac{1}{2}(G_{KL,M} + G_{KM,L} - G_{LM,K}). \tag{35}$$

The gradient of a tensor For a second order mixed tensor we get the components of the gradient,

$$A^i{}_{K;L} = A^i{}_{K,L} + \Gamma^i{}_{ml} x^l{}_{,L} A^m{}_K - \widetilde{\Gamma}^M{}_{KL} A^i{}_M \tag{36}$$

$$A^i{}_{K;l} = A^i{}_{K,l} + \Gamma^i{}_{ml} A^m{}_K - \widetilde{\Gamma}^M{}_{KL} A^i{}_M X^L{}_{,l} \tag{37}$$

A.4 Special tensors

The permutation symbol ϵ_{ijk} is defined as

$$\begin{aligned} \epsilon_{ijk} = \epsilon^{ijk} &= 0 \qquad \text{when any two indices are equal;} \\ &= +1 \quad \text{when } i,j,k \text{ are an evan permutation} \\ &\qquad \text{of the numbers } 1,2,3; \\ &= -1 \quad \text{when } i,j,k \text{ are an odd permutation} \\ &\qquad \text{of the numbers } 1,2,3. \end{aligned}$$

The permutation tensor is defined as

$$\eta_{ijk} = \sqrt{g}\epsilon_{ijk} \qquad \eta^{ijk} = \frac{1}{\sqrt{g}}\epsilon^{ijk} \tag{38}$$

The permutation tensor is defined as

$$\eta_{ijk} = \sqrt{g}\epsilon_{ijk} \qquad \eta^{ijk} = \frac{1}{\sqrt{g}}\epsilon^{ijk} \tag{39}$$

B Examples of *cm*-code

B.1 The code for implementing F

```
cm_Calc_FupDn := proc()
local    mapp,invmapp,mapptable,ndim,i,K,Diffvar,
         Finvtable,Finvmatrix,Ftable,Fmatrix;
   ndim := cmGetMetricDim();
   if cmCheckIfMapping() then
      mapptable := cmGetObjTable(chi(up));
      Diffvar := cmGetRefVar();
      for i to ndim do
        for K to ndim do
      Ftable[i,K]:=diff(mapptable[i],Diffvar[K]);
        od;
      od;
      cmSetObjTable(F(up,Dn),Ftable);
   elif cmCheckIfInvmapping() then
      cmCalc(Finv(Up,dn));
      Finvtable := cmGetObjTable(Finv(Up,dn));
      Finvmatrix := linalg[matrix](ndim, ndim);
      for K to ndim do
         for i to ndim do
      Finvmatrix[K,i] := Finvtable[K,i];
         od;
      od;
      Fmatrix := linalg[inverse](Finvmatrix);
      for K to ndim do
         for i to ndim do
      Ftable[K,i] := Fmatrix[K,i];
         od;
      od;
      cmSetObjTable(F(up,Dn),Ftable);
   else
      print(`You need to load a mapping,`);
      print(`or an inverse mapping,`);
      print(`before calculating this object`);
   fi;
end;
```

B.2 The code for implementing B

```
cm_Calc_Bupup := proc()
local   Gtable,Ftable,Btable,Bmatrix,ndim,
        i,j,K,L,Binvtable,Binvmatrix;
   ndim := cmGetMetricDim();
   if cmCheckIfMapping() then
      cmCalc(F(up,Dn));
      Ftable := cmGetObjTable(F(up,Dn));
      cmCalc(G(Up,Up));
      Gtable := cmGetObjTable(G(Up,Up));
      for i to ndim do
         for j to ndim do
            objtable[i,j] := 0;
            for K to ndim do
               for L to ndim do
      Btable[i,j]:=Btable[i,j]+Gtable[K,L]*Ftable[i,K]*Ftable[j,L];
               od;
            od;
         od;
      od;
   elif cmCheckIfInvmapping() then
      cmCalc(Binv(dn,dn));
      Binvtable := cmGetObjTable(Binv(dn,dn));
      Binvmatrix := linalg[matrix](ndim, ndim);
      for i to ndim do
         for j to ndim do
      Binvmatrix[i,j] := Binvtable[i,j];
         od;
      od;
      Bmatrix := linalg[inverse](Binvmatrix);
      for i to ndim do
         for j to ndim do
      Btable[i,j] := Bmatrix[i,j];
         od;
      od;
   else
      print(`You need to load a mapping,`);
      print(`or an inverse mapping,`);
      print(`before calculating this object`);
   fi;
   cmSetObjTable(B(up,up),Btable);
end;
```

References

[1] Musgrave, P., Pollney, D. and Lake, K.,
1996 www.astro.queensu.ca/ grtensor/GRHome.html

[2] Musgrave, P. and Lake, K.,
Engineering Applications of GRTensorII: Nonlinear Elasticity-Finite Deformations 130.15.26.62/NewDemo/Eng/frame.html

[3] Narasimhan, M.N.L., *Principles of Continuum Mechanics* Wiley, New York 1993.

[4] Truesdell, C. and Noll, W., *The non-linear field theories of mechanics.* Handbuch der Physik III/3. Springer-Verlag Berlin 1965.

[5] Söderholm, L.H., *Mechanics and thermodynamics of contiua, second edition* KTH Department of Mechanics 1997.

[6] Ericksen, J.L., Tensor Fields. In Handbuch der Physik III/1. Springer-Verlag, Berlin 1960.