Toolbox for Grassmann Manifold Computations

Berkant Savas
Department of Mathematics, Linköping University
besav@math.liu.se

October 2, 2008

Abstract

This a description and user guide for an object oriented toolbox written in MATLAB for computations defined on Grassmann manifolds and products of Grassmann manifolds. It implements basic operations as geodesic movement and parallel transport of tangent vectors.

1 Contents of toolbox

There are two classes:

- `@grass` — Contains functions, methods and operations for objects on one Grssmannian.
- `@prodGrass` — Contains the very same functions and methods as `@grass` but for objects defined on a product of Grassmannians. This class is build on top of the `@grass`-class.

Both classes contain the following functions:

File name	Short description
`display.m`	Displays some information about the object.
`get.m`	Gets/extracts various data from the object.
`innerProd.m`	Computes the inner product between two tangent vectors.
`move.m`	Moves the current point and/or parallel transports tangent vectors.
`norm.m`	Computes the norm of a tangent vector.
`set.m`	Sets/initiates various data-fields of the object.

In addition each class has an object constructor file `grass.m` and `prodGrass.m`.

The toolbox implements functions and operations on a Grassmann manifold and a product of Grassmann manifolds described in [2]. It is available at http://www.mai.liu.se/~besav/soft.html. To use the toolbox download the zip-file, unpack it and make sure to add the folder in the "MATLAB search path" (File → Set Path...).

I am assuming the reader has the necessary theoretical knowledge about manifolds and various operations on manifolds. Otherwise consider e.g. [2, 1, 3, 4].

2 Function descriptions

In this section we will give a detailed description of the different files and how they can be used.

2.1 Declaring and constructing objects

A Grassmann object is declared by calling `grass.m`. The object contains the following data structures:

Property name	Description
`dim`	Dimension of the Grassmannian, i.e. if $\mathrm{Gr}(n,r)$ then `dim` = [n r].
`data`	A representation of a point $X \in \mathrm{Gr}(n,r)$.
`base`	A basis for the tangent spaces $X_\perp$ s.t. $[X\ X_\perp]$ is orthogonal.
`proj`	Projection on the tangent space $\mathbb{T}_X$, $\Pi_X = I - XX^\mathsf{T}$.
`tan`	A tangent vector/matrix $\Delta \in \mathbb{T}_X$, it follows that $\Delta^\mathsf{T} X = 0$.
`tan2`	A second tangent $\Delta_2 \in \mathbb{T}_X$. Of course $\Delta_2^\mathsf{T} X = 0$.
`rtan`	A tangent in local coordinates D. If $\Delta \in \mathbb{T}_X$ then $\Delta = X_\perp D$.
`rtan2`	A second tangent in local coordinates D_2.
`svd`	A struct with the thin SVD factors U, Σ, V of the first tangent Δ.
`rsvd`	A struct with the thin SVD factors U, Σ, V of D.

There are three different ways to create a `grass`-object:

```
>> g = grass()
```

This is an empty `grass`-object, no fields are initialized. Given the dimensions n, r of the manifold $\mathrm{Gr}(n,r)$, e.g. $n = 10$ and $r = 3$ we can use

```
>> n = 10; r = 3;
>> g = grass([n,r])
```

If `g` is a `grass`-object one can copy it (with all fields that are initialized) by

```
>> g2 = grass(g)
```

A `prodGrass`-object contains two fields:

Property name	Description
`grass`	A cell array where each entry is a `grass`-object.
`prodSize`	The number of `grass`-objects in the product manifold.

There are two ways to initiate `prodGrass`-objects. The first one inputs the number of `grass`-objects and the second one takes two vectors of equal dimensions and integer values, which specify the dimensions of the Grassmannians, as input arguments. The command

```
>> G = prodGrass(3)
```

declares a `prodGrass`-object with three Grassmannians. No fields of the `grass`-objects are initialized. Alternatively, given $n = [9\ 8]^\mathsf{T}$ and $r = [3\ 4]^\mathsf{T}$ the product manifold $\mathrm{Gr}(9,3) \times \mathrm{Gr}(8,4)$ is initialized simply by

```
>> n = [9,3]'; r = [3,4]';
>> G = prodGrass(n,r)
```

In this case the dimension of the `grass`-objects are initialized accordingly. Throughout this manual we will use lower case `g` to denote a `grass`-object and a capital `G` to denote a `prodGrass`-object.

2.2 Setting variables to the data-fields

A point on Grassmann manifold $\mathrm{Gr}(n,r)$ is represented by an orthonormal matrix X, i.e. $X^\mathsf{T}X = I_r$. A given point, r-dimensional subspace of $\mathbb{R}^n$ has infinitely many representations. Any matrix whose columns span the subspace in question will do. Given a matrix $X \in \mathrm{Gr}(n,r)$, and a `grass`-object `g` the command

```
>> g = set(g,'data',X)
```

initiates the point of `grass`-object `g` to X.

Similarly, if X instead is a cell array containing two orthonormal matrices, e.g. $X_1 \in \mathrm{Gr}(9,3)$, $X_2 \in \mathrm{Gr}(8,4)$ and $X\{1\} = X_1$, $X\{2\} = X_2$ and G already declared `prodGrass`-object, the commands

```
>> X{1} = orth(randn(9,3));
>> X{2} = orth(randn(8,4));
>> G = set(G,'data',X);
```

set the actual points/matrices in the data-fields. Observe that there is no check for orthogonality of the input argument in either case!

Both the basis matrix for the tangent space and the projection matrix onto the tangent space are set by giving the actual point X, e.g. with g a `grass`-object

```
>> g = set(g,'base',X);
>> g = set(g,'proj',X);
>> % or if X is a cell-array and G a prodGrass-object
>> G = set(G,'base',X,'proj',X);
```

To set the a (first) tangent vector T, i.e. $X^\mathsf{T}T = 0$ we use

```
>> g = set(g,'tan',T);
>> % or if T is a cell array with the corresponding tangents
>> G = set(G,'tan',T);
```

Setting the first tangent(s) also sets the `'svd'`-field in the object! This is the intended direction of movement. To set the second tangents simply replace the `'tan'` string with `'tan2'`. To set tangents in local coordinates is similar, just replace `'tan'` with `'rtan'` or `'tan2'` with `'rtan2'`. Also in local coordinates the first tangent matrix is intended to give the direction of movement, and thus sets the `'rsvd'`-field.

Remark. There are no internal checks to ensure that the inputs are indeed tangents when setting the `'tan'`- or `'tan2'`-fields. The situation is different for the `'rtan'` and `'rtan2'` tangents. They will always represent vectors on the tangent plane since they have an associated basis matrix $X_\perp$ which is stored in the `'base'`-field.

2.3 Reading variables from the data-fields

Extraction of data from the fields of a `grass`- or `prodGrass`-object is done trough the `get` function. One field at a time is extracted. The results is stored in a variable or displayed in the command window. If the input object argument is a `grass`-object the results is a matrix, except for the `'svd'` cases where the result will be a struct with three matrices. If the input object is instead a `prodGrass`-object the get function will return a cell array of the same size as the number of Grassmann manifolds. Here are a few examples using the get function used on a `grass`-object `g`.

```
>> X = get(g,'data');
>> P = get(g,'proj');
>> B = get(g,'base');
>> T = get(g,'tan');
>> S = get(g,'svd');
```

The syntax is the same when extracting data from a `prodGrass`-object `G`.

2.4 To move a point and tangent vectors along geodesics

The `move` function implements movement of points and parallel transport of tangents along a geodesic. A step size needs to be specified. The function mirrors the way it is used in optimization algorithms, for example Newton or quasi-Newton methods, for minimization of nonlinear functions defined on a Grassmannian or a product of Grassmannians. The function takes three arguments: (1) a `grass-` or `prodGrass`-object, (2) step size t and (3) one of the following string options `'p'`, `'ptt'` of `'pb'`.

String	Description
`'p'`	Only the point X is moved along the geodesic given by the direction in the `'tan'`-field.
`'ptt'`	The point X as well as tangents (in `'tan'`- and `'tan2'`-fields) are transported.
`'pb'`	The point X and the basis matrix $X_\perp$ are transported.

Here are some explanations for these choices.

- In a Newton method only the point is moved. At the new point the gradient and the Hessian of the objective function is computed. The Newton equations are solved and a direction of movement is obtained. The next iterate is a point along this direction.
- In quasi-Newton methods one does not computed the Hessian at each new point. Instead the gradient and the search direction from the previous point are used to obtain a new direction of movement at the new point. In these case we do need to parallel transport the gradient (stored in the `'tan2'`-field) and the search direction (stored in the `'tan'`-field). This is only done when the algorithm is implemented in global coordinates.
- When implementing a quasi-Newton algorithm in local coordinates the representation of the tangents does not change when they are transported along a geodesic, when considered in suitably transported basis matrix. Thus we move the point and parallel transport the basis matrix.

Here are a few examples to move a point X along the tangent direction stored in the `'tan'`-field. The step length is t.

```
>> g = move(g,t,'p');
>> g = move(g,t,'ptt');
>> % The syntax is the same in the product manifold case.
>> G = move(G,t,'p');
>> G = move(G,t,'pb');
```

The exact computations performed in each of the steps are the following. Let $X \in \mathrm{Gr}(n,r)$ and $\Delta, \Delta_2 \in \mathbb{T}_X$. The geodesic path on from X in the direction of Δ is given by

$$X(t) = \begin{bmatrix} XV & U \end{bmatrix} \begin{bmatrix} \cos \Sigma t \\ \sin \Sigma t \end{bmatrix} V^T \tag{1}$$

where $U\Sigma V^\mathsf{T} = \Delta$ is the thin SVD. Any matrix representation of point on a Grassmannian may be postmultiplied by any orthogonal matrix and this will only change its representation. The subspace and thus the point will be the same. Thus, we may omit V at the end and this is the case when `move` is called with the `'p'` option. The computation becomes simply

$$X(t) = XV \cos(\Sigma t) + U \sin(\Sigma t).$$

When move is called with the `'pb'` or `'ptt'` options the V matrix is present in the computation, i.e. it implements equation (1). In these two cases V is necessary for consistency reasons.

Calling `move` with `'ptt'` also parallel transports the two tangent matrices. The parallel transport computations for Δ_2 are

$$\Delta_2(t) = \left(\begin{bmatrix} XV & U \end{bmatrix} \begin{bmatrix} -\sin \Sigma t \\ \cos \Sigma t \end{bmatrix} U^\mathsf{T} + (I - UU^\mathsf{T}) \right) \Delta_2.$$

Inserting Δ instead of Δ_2 above and using $U\Sigma V^\mathsf{T} = \Delta$ the parallel transport computation of Δ becomes

$$\Delta(t) = \begin{bmatrix} XV & U \end{bmatrix} \begin{bmatrix} -\sin \Sigma t \\ \cos \Sigma t \end{bmatrix} \Sigma V^\mathsf{T}.$$

In addition to this computation the SVD of $\Delta(t)$ is computed and stored in the `'svd'`-field of the object. In this way the SVD of the direction of movement (stored in the `'tan'`-field) will always be up to date.

Calling the move function with the `'pb'` option parallel transports the basis matrix $X_\perp$. The assumptions for this call are that the basis matrix $X_\perp$, and SVD of the local coordinate representation of the first tangent are initialized in the `grass`-object. The performed computation is

$$X_\perp(t) = \begin{bmatrix} X\bar{V} & X_\perp \bar{U} \end{bmatrix} \begin{bmatrix} -\sin \bar{\Sigma} t \\ \cos \bar{\Sigma} t \end{bmatrix} \bar{U}^\mathsf{T} + X_\perp (I - \bar{U}\bar{U}^\mathsf{T}). \tag{2}$$

In the representation of a tangent $\mathbb{T}_X \ni \Delta = X_\perp D$ the left hand side is the global coordinate representation and D in the right hand side is its local coordinate representation. In (2) we use the compact SVD $\bar{U}\bar{\Sigma}\bar{V}^\mathsf{T} = D$.

2.5 Inner product between tangents and norm of tangents

The canonical inner product between tangents $D, F \in \mathbb{T}_X$ on a Grassmann manifold is given by

$$\langle D, F \rangle = \mathrm{tr}(D^\mathsf{T} F),$$

where tr denotes the trace operator. If we have a product of k manifolds then with $D = (D_1, \ldots, D_k)$ and $F = (F_1, \ldots, F_k)$ tangents, i.e. $D_i, F_i \in \mathbb{T}_{X_i}$, the canonical inner product is

$$\langle D, F \rangle = \sum_i \mathrm{tr}(D_i^\mathsf{T} F_i).$$

The induced norm is simply $\langle F, F \rangle$. The function `innerProd` takes a `grass`- or `prodGrass`-object and computes the inner product between the first and second tangents stored in its own fields. Thus if `g` or `G` are given with both tangents initialized the inner product is computed by

```
>> innerProd(g)
>> innerProd(G)
```

Currently the `innerProd`-function implementation only uses global coordinates, i.e. the tangent must be stored in the `'tan'`- and `'tan2'`-fields.

The `norm`-function computes the norm of a tangent in any of the four tangent fields (`'tan'`, `'tan2'`, `'rtan'` and `'rtan2'`).

```
>> norm(g,'tan')
>> norm(g,'rtan2')
>> norm(G,'tan2')
```

3 Complete example using the toolbox

This section illustrates the use of the `grass`-class objects and shows outputs from one run of the code. These commands are available in the `test_grass_class.m` file. Similar operations and computations for product of manifolds are found in the `test_prodGrass_class.m` file.

Initiate variables

```
>> n = 5;
>> r = 2;
>> X  = orth(randn(n,r));  % A point on Gr(n,r)
>> % Global coordinates for two tangents.
>> Tg  = (eye(n) - X*X')*randn(n,r);
>> Tg2 = (eye(n) - X*X')*randn(n,r);
```

Declare the grass-object and initiate some fields.

```
>> g = grass([n,r])
g is a point on a Grassmann manifold of size 5 x 2
>> g = set(g,'data',X);
>> g = set(g,'tan',Tg, 'tan2',Tg2,'base',X);
>> % Get the basis matrix.
>> Xp = get(g,'base');
>> % Local coordinate for the two tangents.
>> Tl  = Xp'*Tg;
>> Tl2 = Xp'*Tg2;
```

Tests and operations.

```
>> get(g,'data')
ans =
   0.471879991188995   0.511406753625131
  -0.522794037473823  -0.035856913416763
   0.478373679563317   0.326917166263619
  -0.330765624120784   0.496771020047634
  -0.407146647902613   0.619290831637144
>> % The svd of first tangent is already set!
>> s = get(g,'svd')
s =
    U: [5x2 double]
    S: [2x2 double]
    V: [2x2 double]
>> % Verify this is the case
>> norm( s.U*s.S*s.V' - get(g,'tan') )
ans =
    4.884063998313457e-016
>> % Compute the norm of first and second tangent.
>> norm(g,'tan')
ans =
   8.622633600270721
>> norm(g,'tan2')
ans =
    2.667500708367026
>> % norm(g,'rtan2')          % rtan2-field not yet initialized.
>> g = set(g,'rtan2',Tl2);
>> norm(g,'rtan2')
ans =
   2.667500708367026
```

Compute the inner product between the first and second tangent.

```
>> innerProd(g)
ans =
  -2.685155976514037
```

Move operations.

```
>> % t is the step length
>> t = 0.5;
>> % Move just the point X.
>> g2 = move(g,t,'p');
>> X2 = get(g2,'data')
X2 =
```

```
 -0.234787222411819   0.402948778889981
 -0.044633706376310  -0.757534064203116
 -0.005765326543034  -0.010426742621727
 -0.810316638210536  -0.333014180145568
  0.535010746897933  -0.390854527955425
>> % X2 is a point on the manifold!
>> X2'*X2
ans =
   1.000000000000000  -0.000000000000000
  -0.000000000000000   1.000000000000000
>> % But tangents in g2 are not transported!
>> X2'*get(g2,'tan')
ans =
  -0.550974682913990  -2.637572909385948
   0.502050139785949  -0.104875552668566
>> X2'*get(g2,'tan2')
ans =
   0.160640294264665   0.773091619169026
  -0.088550365497921   0.681260368210154
>> % Nor the basis matrix is transproted.
>> X2'*get(g2,'base')
ans =
   0.130368427968324  -0.693861388940499   0.680631813561688
  -0.349074947549363   0.210248948941966   0.281197522768956
```

Now move the point ant both tangents.

```
>> g3 = move(g,t,'ptt');
>> X3 = get(g3,'data')
X3 =
  0.442444175686822   0.147431033427812
 -0.732401077354089   0.198591762325498
 -0.009027533289630   0.007775576798680
 -0.160283623960261   0.861290112696654
 -0.491995403255202  -0.443784051645948
>> % Now tangents are transported and of courese they
>> % are orthogonal to the current point.
>> X3'*get(g3,'tan')
ans =
  1.0e-015 *
  -0.055511151231258  -0.444089209850063
   0.069388939039072   0.777156117237609
>> X3'*get(g3,'tan2')
ans =
  1.0e-015 *
   0.144849410244063   0.111022302462516
   0.068521577301084   0.111022302462516
>> % Compute the inner product again, and compare with
>> % the computation at the previous point: innerProd(g)
>> innerProd(g3)
ans =
  -2.685155976514037
```

Move the point and the basis matrix.

```
>> % First set the first tangent (direction of movement)
>> % in local coordinates. The rsvd-field is set automatically.
```

```
>> g = set(g,'rtan',Tl);
>> g4 = move(g,t,'pb');
>> X4 = get(g4,'data');
>> X4'*get(g4,'base')
ans =
  1.0e-015 *
   0.062450045135165  -0.235922392732846                    0
   0.006938893903907  -0.027755575615629  -0.111022302462516
>> % The matrices X2,X3,X4 represent the same point/subspce
>> subspace(X2,X3)
ans =
   4.714194175523463e-016
>> subspace(X2,X4)
ans =
   4.951428291383124e-016
>> % But X2 is different from X3 and X4.
X2-X3
ans =
  -0.677231398098641   0.255517745462169
   0.687767370977779  -0.956125826528614
   0.003262206746597  -0.018202319420407
  -0.650033014250275  -1.194304292842222
   1.027006150153135   0.052929523690523
```

References

[1] P.-A. Absil, R. Mahony, and R. Sepulchre. *Optimization Algorithms on Matrix Manifolds*. Princeton University Press, Princeton, NJ, January 2008.

[2] A. Edelman, T.A. Arias, and S.T. Steven. The geometry of algorithms with orthogonality constraints. *SIAM J. Matrix Anal. Appl.*, 20(2):303–353, 1999.

[3] L. Eldén and B. Savas. A Newton-Grassmann method for computing the best multilinear rank-(r_1, r_2, r_3) approximation of a tensor. Technical Report LITH-MAT-R-2007-6-SE, Department of Mathematics, Linköpings Universitet, 2007.

[4] B. Savas and L.-H. Lim. Best multilinear rank approximation of tensors with quasi-Newton methods on Grassmannians. Technical Report LiTH-MAT-R–2008-01–SE, Department of Mathematics, Linkoping University, April 2008.