Guide to the Mathematica programs used in the paper arXiv:1609.01946**(ver.3)**

December 21, 2016

1 Introduction

This is a guide to the Mathematica programs used in the paper titled “Equation of motion of canonical tensor model and Hamilton-Jacobi equation of general relativity”. To carry out the complicated computations in the paper, we heavily used the “xTensor” package for tensorial computations, instead of doing them by hand. Since at present it is hard to reproduce the results without such machine powers, we keep the programs for future possible correspondence to the paper (We hope, however, that the derivations would be much simplified enough to do by hand in future study.). The purpose of this guide is not for any researchers to understand the programs. Rather, this is a brief reminding note about the programs. Or, really interested researchers may find the guide helpful to use the programs for further study.

2 The environment

We do not know whether the programs run in the same manner in any environments. If the programs do not work properly, difference of environment would be the main suspect. Ours is

- xTensor version: 1.1.2 (2015/8/23)
- Mathematica version: 10.0.2.0
- Machine & OS: MacBook Pro (Retina, Mid 2012), Mac OS X 10.11.6

3 How to run

Install the xTensor package (freely downloadable).

Each program contains three cells. Run the first cell: Quit[] will set up a new kernel to avoid malfunctions caused by former definitions. Then, run the next cell, which is the main. The third cell is the output obtained in our environment. This is for assurance, in case the programs do not run properly in your environment.

4 What is done in each program

4.1 Parameters and variables

The parameter ep is used to count the number of derivatives in derivative expansions. For example, a two-derivative term is expressed like

```
epˆ 2 CD[-a1][f[]]CD[-a2][f[]]
```

accompanied with epˆ2.

We set a1-a6,b1-b6,c1-c6 as indices of tensors.

Dimension (spatial) is denoted by d.

As variables, we try to use expressions with similarity with those in the paper. For example,

$$\begin{aligned}&\beta : \mathrm{be[]},\\ &\beta^{\mu\nu} : \mathrm{be2[a1,a2]},\\ &\beta^{\mu\nu\rho} : \mathrm{be3[a1,a2,a3]},\\ &\beta^{\mu\nu,\rho\sigma} : \mathrm{be22[a1,a2,a3,a4]}.\end{aligned} \tag{1}$$

In the computations of two successive infinitesimal time evolutions, we also use w and w2 to represent the output:

$$\begin{aligned}&n_1\nabla_\mu n_2 - n_2\nabla_\mu n_1 : \mathrm{w[-a1]},\\ &n_1\nabla_\mu\nabla_\nu n_2 - n_2\nabla_\mu\nabla_\nu n_1 : \mathrm{w2[-a1,-a2]}.\end{aligned} \tag{2}$$

4.2 EOMinBeta.nb

This program derives the equation of motion of CTM with the variables $\beta, \beta^{\mu\nu}, \beta^{\mu\nu\rho}, \beta^{\mu\nu,\rho\sigma}$. The curvature term is computed in another program. The corresponding equation is (24) in the paper. The main output is the list of EOM. Then, they are compared with the symmetric expressions in the paper. The output of the comparison should appear as 0's.

4.3 Curvature.nb

This program computes the curvature term in the equation of motion. The corresponding equation is (115).

4.4 ComBeta.nb

This program computes the commutation of two successive infinitesimal time evolutions of $\beta, \beta^{\mu\nu}$. Then, it compares the results with the diffeomorphism transformations of them. The corresponding equations are (29) and (30).

4.5 Beta2Metric.nb

This program is to rewrite the right-hand sides of EOM by relating the background metric tensor with the fields as $\frac{g^{\mu\nu}}{\sqrt{g}} = \beta\beta^{\mu\nu}$. The corresponding equation is (47).

4.6 BetawithMetric.nb

This program obtains the full EOM in the case that the background metric is given by $\frac{g^{\mu\nu}}{\sqrt{g}} = \beta\beta^{\mu\nu}$. Then, it computes the two successive infinitesimal time evolution for the consistency check. The corresponding equations are (52), (55) and (56).

4.7 Determineh.nb

This program obtains the EOM (66) (n not replaced.). It also obtains (65) for the correction h.

4.8 EOMNoDerN.nb

This program first checks the algebraic relation (67) with (68) for the consistency check of the EOM (66) (This is computed with free parameters x_i, but this is the same thing.). Then, it obtains the equations (117) of x_i for the vanishing condition of the derivative terms of n, and solves for $x_1, \cdots, x_6$ to obtain the EOM, and compare it with the expression (72) of the paper.

4.9 GR.nb

This program computes the potential and EOM of the coupled system of GR and a scalar field in the Hamilton-Jacobi formalism. The corresponding equations are (87) and (90).

4.10 Matching.nb

It first obtains the equations of the matching condition between CTM and GR, and obtain the solution (91). Then, putting the explicit value of c_3, it obtains V to check the other conditions (87).

5 Possible run-time errors in other environments

The programs contain some command lines which are sensitive to expressions of former intermediate results. Therefore, we are not sure whether our programs work properly in other environments, since mathematically same former intermediate results may be expressed differently in other environments. For instance, if a dummy(contracted) index a1 of a tensor in an intermediate result in our original computation appeared as b1 in another environment for some reason (possibly by the difference of the version of xTensor for instance), then the program would not work properly. There are two main kinds of places where such sensitive command lines exist in our programs:

- Since the indices of the derivatives and the metric tensors must respectively be the lower and upper indices throughout the computations, to corretly deal with the degrees of freedom, there are some command lines which properly restore the cases after being changed by Simplification. For example, we apply replacements to intermediate results like

  an intermediate result /. { g[-a1,-a2] → g[a1,a2], CD[a1][x_] → CD[-a1][x], CD[a2][x_] → CD[-a2][x] }

  This will not work properly, if the indices of the intermediate result are not a1 and a2 for some reason.

- We use Module to define a function with some dummy(contracted) variables to avoid multiple appearances of same contracted indices in later computations. For example, when we want to define a function be[] with two dummy variables, we do

  fbe[a1_ ,a2_]= an intermediate result with dummy variables a1 and a2
  be[]:=Module[{a1,a2},fbe[a1,a2]]

  This defines be[] by local variables for the dummy indices a1 and a2 (This looks awkward from a mathematical point of view, because dummy variables appear as arguments of the function fbe, but works correctly because dummy indices are not really summed over in Mathematica.). However, if the dummy variables in the intermediate result were not a1 and a2, differently from our original computation, be[] would not be defined by local variables, but by global variables. Then, in later computations, an error would occur, because of multiple appearances of same contracted indices.

  To check the absence of the above kinds of errors, we put some self-checking lines in the programs. For example, when you run the programs, you will see output sentences like

  Check ({a1,a2}) = {a1,a2}

  If the right-hand side is different from the content in the brackets on the left-hand side, then the program is not running properly. In such a case, you would have to look into the intermediate result, and change the program properly by replacing the erroneous indices. In general, this should be a minor straightforward modification of the program.