The Theory and Practice of Cosmological Perturbations

Part II – Computer Assisted Computation

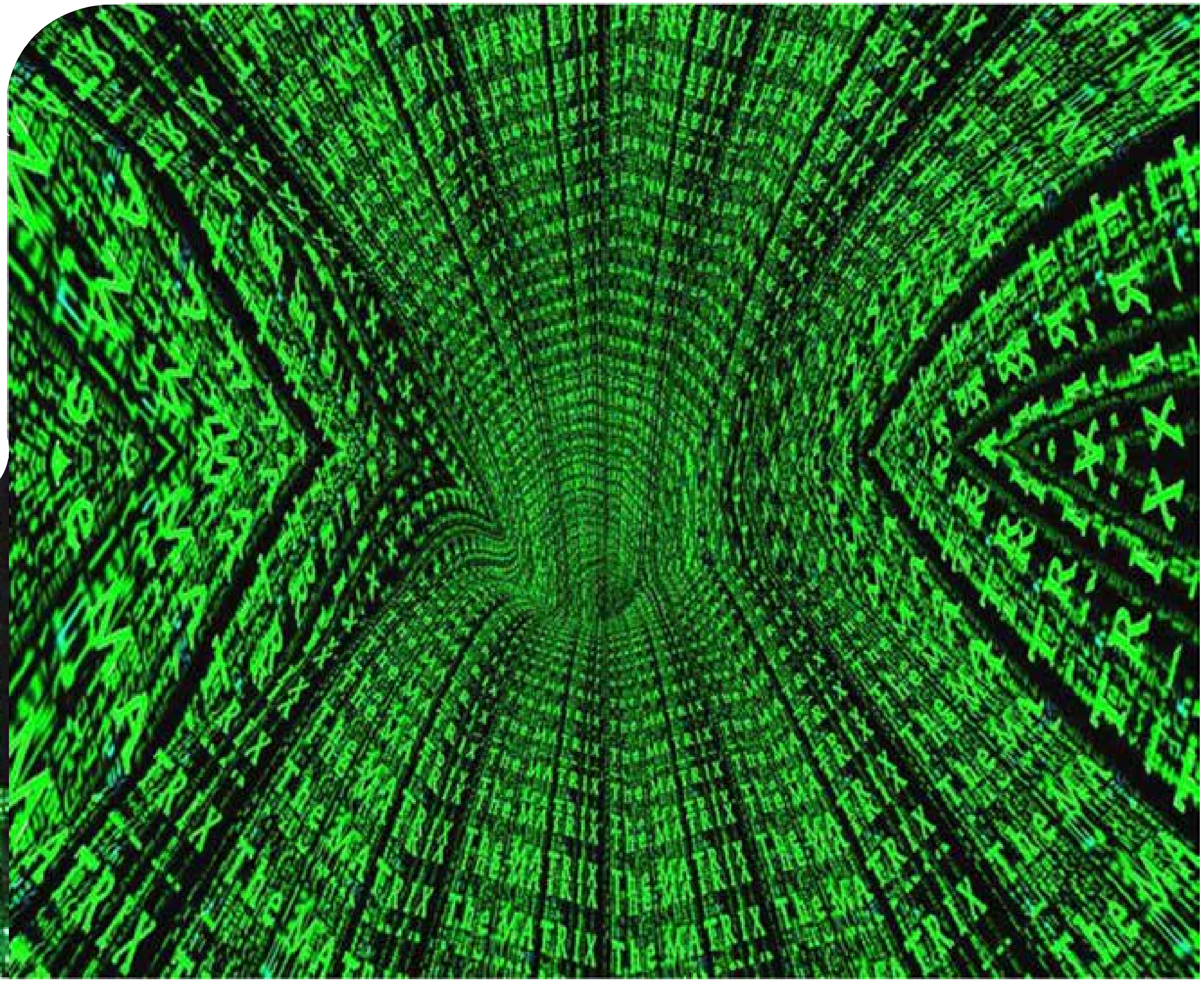
Yi Wang (王一)

The Hong Kong University of Science and Technology

Why Computer Assisted Computation?

I am a physicist,

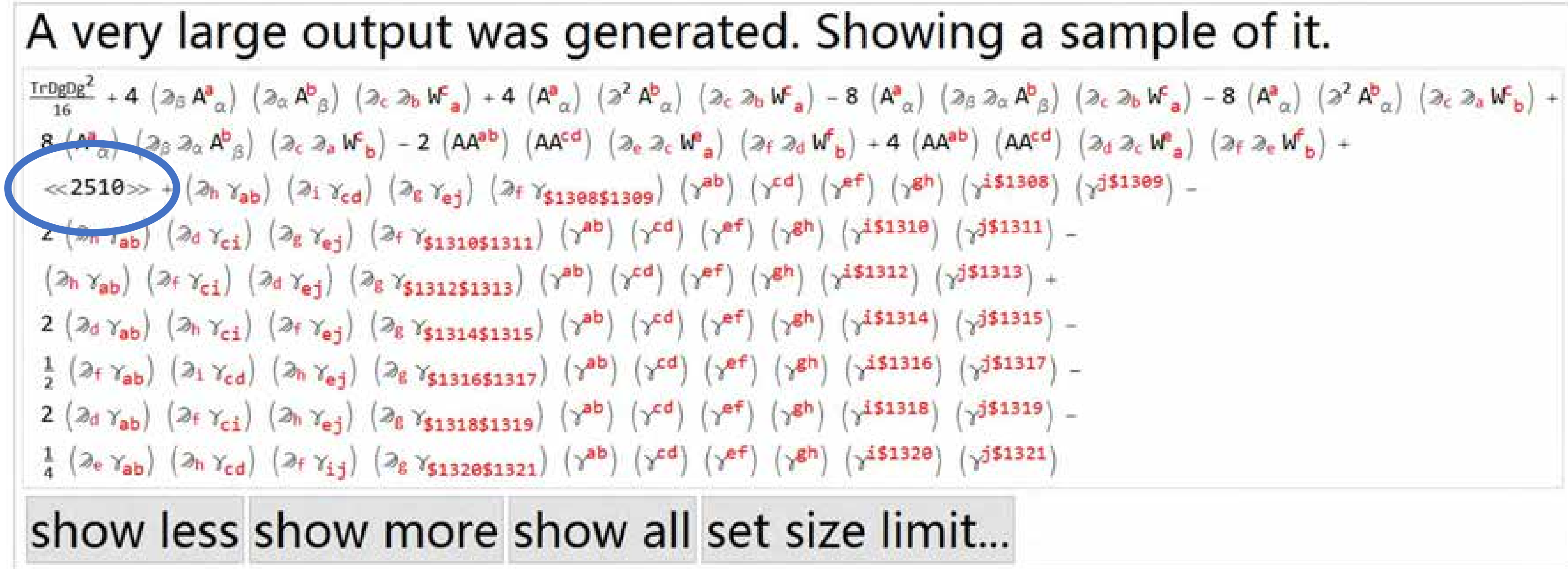
not a technician…




And I work even harder than him!

Why Computer Assisted Computation?

GR is complicated. For example:



I am not smart enough to simplify the length of every calculation.

Example: Cosmological Perturbations

$$ds^2 = -N^2 dt^2 + h_{ij}(N^i dt + dx^i)(N^j dt + dx^j)$$

$$N \equiv 1 + \alpha \qquad N_i = \partial_i \beta + b_i$$

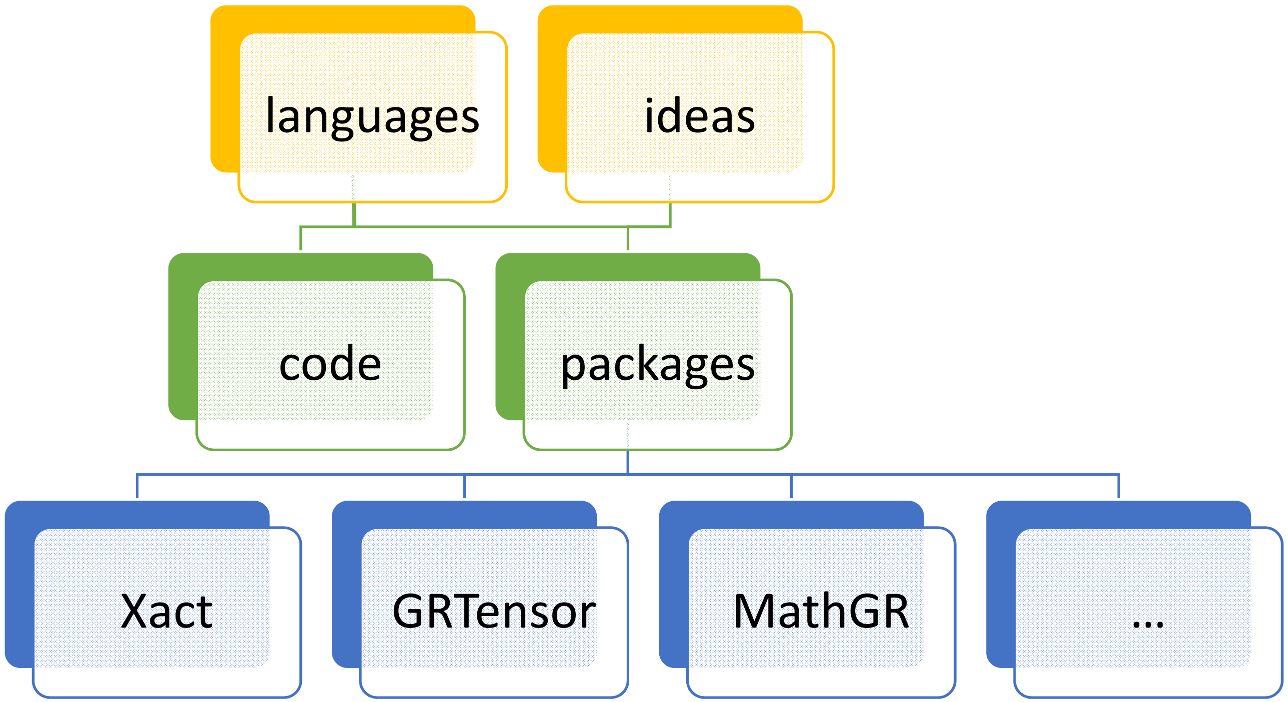
$$h_{ij} = e^{2\zeta} a^2 \left[\delta_{ij} + \gamma_{ij} + \cancel{\mathcal{D}_i \mathcal{D}_j E} + \cancel{\mathcal{D}_i F_j} + \cancel{\mathcal{D}_j F_i}\right]$$

Calculate the Ricci scalar to 3rd order (even in AdM form)


Sometimes we need
brute force calculations


大力出奇迹

How to link cosmology to code?



Languages for Symbolic Calculation

Mathematica

Python (Sympy, Sage)

Maple

Languages: Mathematica

Mathematica

Pros:

- Easy to start
- Popularity
- Interface
- Vocabulary
- Pattern + functional prog
- Intuitive math

Cons:

- Non-free
- Pattern + functional prog

Languages: Mathematica

```
BeginPackage["MathGR`frwadm`", {"MathGR`tensor`", "MathGR`decomp`", "MathGR`gr`", "MathGR`util`", "MathGR`ibp`"}]

DeclareIdx[{UP, DN}, DefaultDim, LatinIdx]

PdT[Mp,_]:=0
PdT[a|H|ε|η, PdVars[_DN, ___]]:=0
SimpHook = {DefaultDim->3, Pd[a, DE@0]->a*H, PdT[a, PdVars[DE@0,DE@0]] -> a H^2 - a H^2 ε,
    Pd[H, DE@0]->-ε*H*H, PdT[H, PdVars[DE@0,DE@0]] -> 2 H^3 ε^2 - H^3 ε η,
    PdT[H, PdVars[DE@0,DE@0,DE@0]] -> -6 H^4 ε^3 + 7 H^4 ε^2 η - H^4 ε η^2 - H^4 ε η η2,
    Pd[ε, DE@0]->H*ε*η, Pd[η, DE@0] -> H*η2*η, Pd[η2, DE@0] -> H*η3*η2    }
LapseN = 1 + Eps * α
ShiftN[DN@i_] := Eps * Pd[β, DN@i] + Eps * b[DN@i]
PdT[b[DN@i_], PdVars[DN@i_, ___]]:= 0
b /: b[DN@i_] k[DN@i_]:= 0 (* Above expression in momentum space. *)
Sqrtg:= LapseN*Exp[3*Eps*ζ] * a^3

UseMetric[h]
h[DN@i_, DN@j_]:= a * a * Exp[2*Eps*ζ] * Dta[DN@i, DN@j]
h[UP@i_, UP@j_]:= Exp[-2*Eps*ζ] * Dta[DN@i, DN@j] /a /a

(* 4d metric is used to be decomposed and be replaced. *)
DecompHook = {
    g[DN@i_, DN@j_]:> h[DN@i, DN@j],
    g[DE@0, DE@0]:> (-LapseN^2 + h[UP@#1, UP@#2]ShiftN[DN@#1]ShiftN[DN@#2] &@Uq[2]),
    g[DE@0, DN@i_]:> ShiftN[DN@i],
    g[UP@i_, UP@j_]:> (h[UP@i, UP@j] - ShiftN[DN@#1]ShiftN[DN@#2]h[UP@#1, UP@i]h[UP@#2, UP@j]/LapseN^2 &@Uq[2]),
    g[UE@0, UE@0]:> -1/LapseN^2,
    g[UE@0, UP@i_]:> (h[UP@i, UP@#]ShiftN[DN@#]/LapseN^2 &@Uq[1])}
```

Languages: Mathematica

Insert the constraints

```
s2Solved = s2 /. conSol[[1]]
```

$$-27\, a^3 H^2 \zeta^2 + a\, k^2 \zeta^2 + 9\, a^3 H^2 \epsilon\, \zeta^2 - 18\, a^3 H\, \zeta\, \dot{\zeta} + \frac{2\, a\, k^2 \zeta\, \dot{\zeta}}{H} + a^3 \epsilon\, \dot{\zeta}^2$$

Integration by parts (Here PdHold[f, DE[0]] is the temporal boundary term $\dot{f}$)

```
s2SolvedIBP = s2Solved // Ibp[#, "Rank" → IbpStd2] &
```

$$-a\, k^2 \epsilon\, \zeta^2 + \text{PdHold}\left[-9\, a^3 H \zeta^2 + \frac{a\, k^2 \zeta^2}{H},\ \text{DE}[0]\right] + a^3 \epsilon\, \dot{\zeta}^2$$

Set the temporal boundary term to zero, get final result

```
s2Final = s2SolvedIBP /. PdHold[__] → 0
```

$$-a\, k^2 \epsilon\, \zeta^2 + a^3 \epsilon\, \dot{\zeta}^2$$

```
ToTeX[s2Final]
Hint: set ToTexTemplate = False to output equation only.
%Generated by MathGR/typeset.m, Tue 2 Sep 2014 15:00:12.
\documentclass{revtex4}
\usepackage{breqn}
\begin{document}
\begin{dmath}
-a k^2 \epsilon  \zeta ^2
 + a^3 \epsilon  \dot\zeta {}^2
\end{dmath}
\end{document}
```

Languages: the Python Family

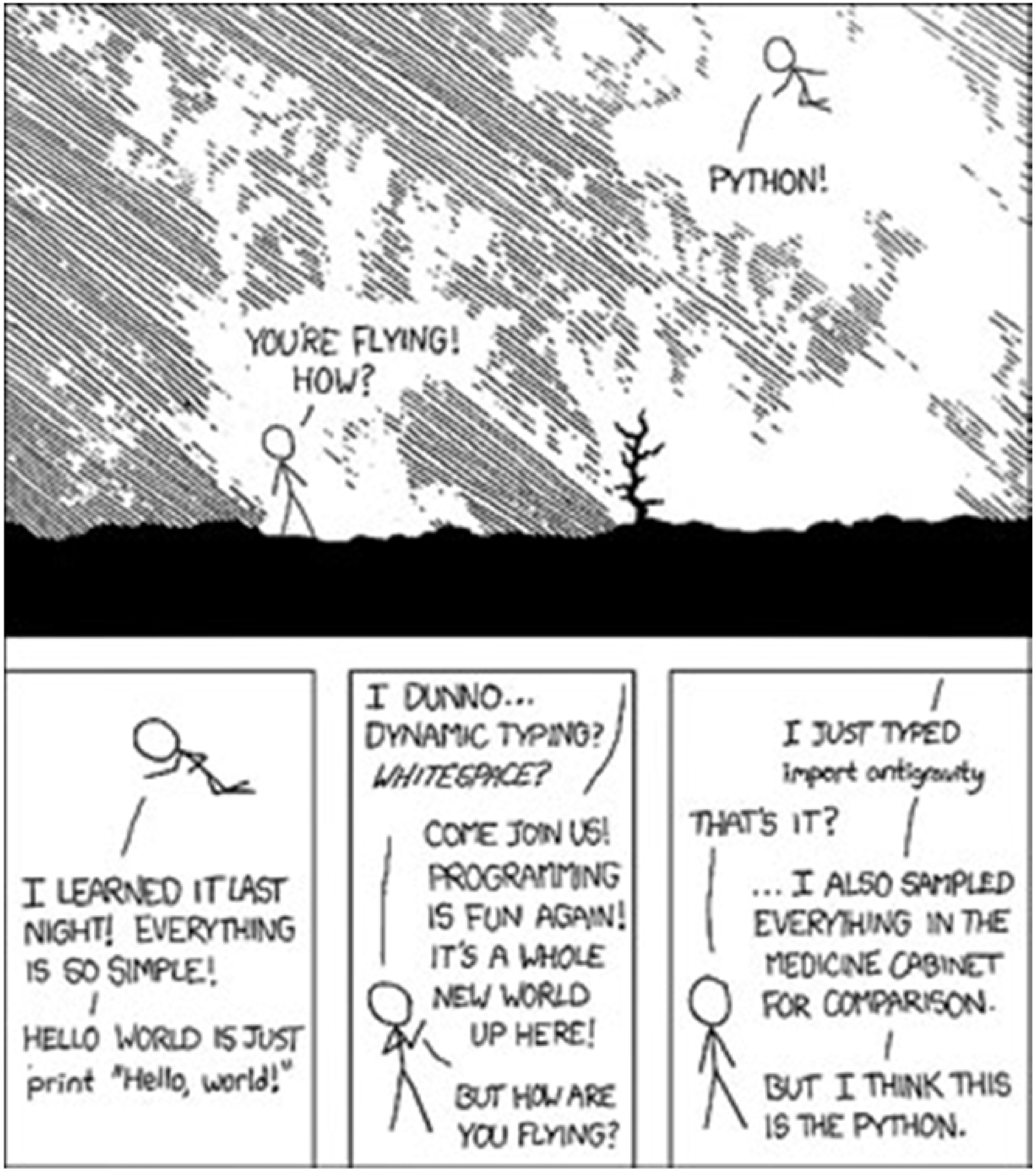
Python (Sympy, Sage)

Pros:

- Free and Open Source
- Great Popularity
- Multiple programming paradigm
- Can fly

Cons:

- Interface (even with Jupyer Notebook)
- Symbolic relatively weak



cartoons

about Python

Languages: the Python Family

models.py

```python
from sage.all import *

import base
import utils
import pert

SAVE_CAL_PATH = '/home/wangyi/Dropbox/work/tools/grpy/save/'


class FRW_full():

    def init_vars(self):
        self.var_str = 'Mp k '
        for x in self.coord:
            self.var_str += str(x) + ' '
        self.bg_fun_list = ['a']

        # perturbation vars
        if self.pert_order == 0:
            self.epsilon = 0
            self.pert_fun_list = []
        else:
            self.epsilon = var('epsilon')
            self.var_str += 'epsilon '
            self.pert_fun_list = ['psi', 'alpha', 'beta', 'delta_phi']

        if self.pert_order == 2:
            self.psi_k = function('psi_k', self.coord[0])
            self.alpha_k = function('alpha_k', self.coord[0])
            self.beta_k = function('beta_k', self.coord[0])
            self.delta_phi_k = function('delta_phi_k', self.coord[0])
```

```python
# define gravity part
if self.gauge == 'flat':
    print("Chosen flat gauge.")
    self.psi = 0
else:
    print("Spatial curvature has perturbation psi.")
    self.psi = function('psi', *self.coord)

self.alpha = function('alpha', *self.coord)
self.beta = function('beta', *self.coord)

N0 = 1 + self.epsilon * self.alpha
N1 = self.epsilon * diff(self.beta,self.coord[1])
N2 = self.epsilon * diff(self.beta,self.coord[2])
N3 = self.epsilon * diff(self.beta,self.coord[3])
AA = self.a**2 * exp(2*self.epsilon*self.psi)
g00 = -N0**2 + (N1**2 + N2**2 + N3**2)/AA

self.metric = [[ g00 ,  N1  ,  N2  ,  N3 ],
               [  N1 ,  AA  ,  0   ,  0  ],
               [  N2 ,  0   ,  AA  ,  0  ],
               [  N3 ,  0   ,  0   ,  AA ]]


if self.matter!=None and self.matter[0]=='scalar':
    # define scalar part
    self.phi0 = function(self.matter[1], self.coord[0])
    self.bg_fun_list.append(self.matter[1])
    if self.gauge == 'uniform_inflaton':
        print('Chosen uniform inflaton gauge.')
```

Languages: Maple

Maple

Maple is very similar to Mathematica.

In my opinion weaker in many senses.

Unless you use GRTensor

http://grtensor.phy.queensu.ca/

Languages: Maple

```
linpert:=proc(gr_obj)
    local new_obj,c0,c1:
    readlib(coeftayl):
    c0:=coeftayl(gr_obj,epsilon=0,0):
    c1:=coeftayl(gr_obj,epsilon=0,1):
    new_obj:=c0+c1*epsilon:
    return(new_obj):
end:

quadpert:=proc(gr_obj)
    local new_obj,c0,c1,c2:
    readlib(coeftayl):
    c0:=coeftayl(gr_obj,epsilon=0,0):
    c1:=coeftayl(gr_obj,epsilon=0,1):
    c2:=coeftayl(gr_obj,epsilon=0,2):
    new_obj:=c0+c1*epsilon+c2*epsilon^2:
    return(new_obj):
end:

subfrw:=proc(gr_obj)
    local
    new_obj,rule1,rule2,rule3,rule4,rule5,rule6,rule7,rule8,rule9,rule10,algrule1,al
    lgrule9,algrule10,algrule11,algrule12,algrule13,algrule14,temp4,listOfSub:
    new_obj:=gr_obj:

    algrule1 := diff(beta(x, y, z, t), x, x, x) =
        (pibpip2b(x,y,z,t)
            -(diff(beta(x, y, z, t), x))*(diff(beta(x, y, z, t), x, y, y)+diff(beta(x, y, z, t), x, z, z))
            -(diff(beta(x, y, z, t), y))*(diff(beta(x, y, z, t), y, x, x)+diff(beta(x, y, z, t), y, z, z)+di
            -(diff(beta(x, y, z, t), z))*(diff(beta(x, y, z, t), z, x, x)+diff(beta(x, y, z, t), z, y, y)+di
            z)))/(diff(beta(x, y, z, t), x)):
```

```
Ndim_ := 4:

x1_ := x:
x2_ := y:
x3_ := z:
x4_ := t:

sig_ := -2:

g11_ := -a(t)^2:
g14_ := -epsilon*diff(beta(x,y,z,t),x):
g22_ := -a(t)^2:
g24_ := -epsilon*diff(beta(x,y,z,t),y):
g33_ := -a(t)^2:
g34_ := -epsilon*diff(beta(x,y,z,t),z):
g44_ := (1+epsilon*alpha(x,y,z,t))^2-epsi

complex_ :={}:

constraint_ := {epsilon^3=0,epsilon^4=0,e
```

Languages: Maple

```
> grtensor( ) :
                    GRTensorII Version 1.79 (R4)
                          6 February 2001
        Developed by Peter Musgrave, Denis Pollney and Kayll Lake
                 Copyright 1994-2001 by the authors.
     Latest version available from: http://grtensor.phy.queensu.ca/
            Defaults read from E:\Grtii\lib/grtensor.ini          (1.1)
> qload(aFRWflatm) :
Calculated ds for aFRWflatm (0.000000 sec.)
                    Default spacetime = aFRWflatm
                    For the aFRWflatm spacetime:
                             Coordinates
                               x(up)
```

$$x^{a} = \left[\, x \;\; y \;\; z \;\; t \,\right]$$

Line element

$$ds^2 = -a(t)^2 \; d\,x^2 - 2\,\varepsilon \left(\frac{\partial}{\partial x}\,\beta(x, y, z, t)\right) d\,x \; d\,t - a(t)^2 \; d\,y^2 - 2\,\varepsilon \left(\frac{\partial}{\partial y}\,\beta(x, y, z, t)\right) d\,y \; d\,t - a(t)^2 \; d\,z^2 - 2\,\varepsilon \left(\frac{\partial}{\partial z}\,\beta(x, y, z, t)\right) d\,z \; d\,t + \left((1 + \varepsilon\,\alpha(x, y, z, t))^2 - \frac{1}{a(t)^2}\left(\varepsilon^2 \left(\left(\frac{\partial}{\partial x}\,\beta(x, y, z, t)\right)^2 + \left(\frac{\partial}{\partial y}\,\beta(x, y, z, t)\right)^2 + \left(\frac{\partial}{\partial z}\,\beta(x, y, z, t)\right)^2 \right)\right)\right) d\,t^2$$

$$Constraints = \left[\varepsilon^3 = 0,\ \varepsilon^4 = 0,\ \varepsilon^5 = 0,\ \varepsilon^6 = 0,\ \varepsilon^7 = 0,\ \varepsilon^8 = 0,\ \varepsilon^9 = 0,\ \varepsilon^{10} = 0\right] \qquad (1.2)$$

```
> read `E:\\My Dropbox\\Ginflation\\frwutils.mpl` :
>
```

Languages for Symbolic Calculation

My personal pick: Mathematica

Languages for Symbolic Calculation

Key points to master

- Vocabulary

```
In[4]:=
    Names["System`*"] // Length
Out[4]=
    5883
```

- Rule based programming

```
{-1, 1, 2, 3} /. a_?Positive :> f[a]
```

- Functional programming (LISP-like)

```
sum = 0;
Do[sum = sum + i, {i, 1, 100}]
sum
Total@Range@100
Plus @@ Range@100
Fold[Plus, 0, Range@100]
```

- Built and import tools

General Ideas about Symbolic GR

Symbolic GR is all about its indices.

How to implement symmetries?

$$\Gamma^{\lambda}{}_{\mu\nu} = \frac{1}{2}\, g^{\lambda\sigma}\partial_{\mu} g_{\overbrace{\sigma\nu}} + \cdots$$

How to treat the dummies?

How to treat the free indices?

Indices Method 1: Explicit Indices

$$\Gamma^{\lambda}{}_{\mu\nu} = \frac{1}{2} g^{\lambda\sigma} \partial_{\mu} g_{\sigma\nu} + \cdots$$

How to treat the dummies?

How to treat the free indices?

Sum[…, $\{\sigma, 1, 4\}$]

Use a multi-dim list to store all results:

$\Gamma^{\lambda}{}_{\mu\nu}$: $\{\{\{\Gamma^{1}{}_{11}\}, \ldots\}, \ldots\}$

Note: Mathematica list start with 1, can use 4 as time.

Indices Method 1: Explicit Indices

$$\Gamma^{\lambda}{}_{\mu\nu} = \frac{1}{2} g^{\lambda\sigma} \partial_{\mu} g_{\sigma\nu} + \cdots$$

```
coord = {x, y, z, t};
d = Length[coord];
g = {
    {a[t], 0, 0, 0},
    {0, a[t], 0, 0},
    {0, 0, a[t], 0},
    {0, 0, 0, 1}};
gInverse = Inverse[g];
```

```
Γ = 1/2 Table[
  Sum[
   gInverse[[λ, σ]]
    (D[g[[σ, ν]], coord[[μ]]]
      + D[g[[σ, μ]], coord[[ν]]]
      - D[g[[μ, ν]], coord[[σ]]]),
   {σ, 1, d}],
  {λ, 1, d}, {μ, 1, d}, {ν, 1, d}];
```

```
In[6]:=
   Γ[[1, 1, 4]]
Out[6]=
   a′[t]
   ------
   2 a[t]
```

Working code can be found at James Hartle's website:

http://web.physics.ucsb.edu/~gravitybook/mathematica.html

Indices Method 1: Explicit Indices

Pros:

+ Very simple code, intuitive

Cons:

+ Very lengthy result
  - Every free indices -> table
  - Every dummy indices -> sum
+ Need to keep track of upper/lower indices ourselves

Indices Method 1: Explicit Indices

$a_{\mu\nu} b^{\nu\lambda}$

```
result = Table[Sum[a[μ, ν] b[ν, λ], {ν, 1, 4}], {μ, 1, 4}, {λ, 1, 4}]
Out[11]=
{{a[1, 1] b[1, 1] + a[1, 2] b[2, 1] + a[1, 3] b[3, 1] + a[1, 4] b[4, 1],
  a[1, 1] b[1, 2] + a[1, 2] b[2, 2] + a[1, 3] b[3, 2] + a[1, 4] b[4, 2],
  a[1, 1] b[1, 3] + a[1, 2] b[2, 3] + a[1, 3] b[3, 3] + a[1, 4] b[4, 3],
  a[1, 1] b[1, 4] + a[1, 2] b[2, 4] + a[1, 3] b[3, 4] + a[1, 4] b[4, 4]},
 {a[2, 1] b[1, 1] + a[2, 2] b[2, 1] + a[2, 3] b[3, 1] + a[2, 4] b[4, 1],
  a[2, 1] b[1, 2] + a[2, 2] b[2, 2] + a[2, 3] b[3, 2] + a[2, 4] b[4, 2],
  a[2, 1] b[1, 3] + a[2, 2] b[2, 3] + a[2, 3] b[3, 3] + a[2, 4] b[4, 3],
  a[2, 1] b[1, 4] + a[2, 2] b[2, 4] + a[2, 3] b[3, 4] + a[2, 4] b[4, 4]},
 {a[3, 1] b[1, 1] + a[3, 2] b[2, 1] + a[3, 3] b[3, 1] + a[3, 4] b[4, 1],
  a[3, 1] b[1, 2] + a[3, 2] b[2, 2] + a[3, 3] b[3, 2] + a[3, 4] b[4, 2],
  a[3, 1] b[1, 3] + a[3, 2] b[2, 3] + a[3, 3] b[3, 3] + a[3, 4] b[4, 3],
  a[3, 1] b[1, 4] + a[3, 2] b[2, 4] + a[3, 3] b[3, 4] + a[3, 4] b[4, 4]},
 {a[4, 1] b[1, 1] + a[4, 2] b[2, 1] + a[4, 3] b[3, 1] + a[4, 4] b[4, 1],
  a[4, 1] b[1, 2] + a[4, 2] b[2, 2] + a[4, 3] b[3, 2] + a[4, 4] b[4, 2],
  a[4, 1] b[1, 3] + a[4, 2] b[2, 3] + a[4, 3] b[3, 3] + a[4, 4] b[4, 3],
  a[4, 1] b[1, 4] + a[4, 2] b[2, 4] + a[4, 3] b[3, 4] + a[4, 4] b[4, 4]}}
```

Indices Method 1: Explicit Indices

One has to recover the compact form oneself.

Trick to make the result less crazy: apply rules to get back the dummy.

```
result /. a[μ_, 1] b[1, λ_] :→ a[μ, ν] b[ν, λ] - a[μ, 2] b[2, λ] - a[μ, 3] b[3, λ] - a[μ, 4] b[4, λ]
3]=
{{a[1, ν] b[ν, 1], a[1, ν] b[ν, 2], a[1, ν] b[ν, 3], a[1, ν] b[ν, 4]},
 {a[2, ν] b[ν, 1], a[2, ν] b[ν, 2], a[2, ν] b[ν, 3], a[2, ν] b[ν, 4]},
 {a[3, ν] b[ν, 1], a[3, ν] b[ν, 2], a[3, ν] b[ν, 3], a[3, ν] b[ν, 4]},
 {a[4, ν] b[ν, 1], a[4, ν] b[ν, 2], a[4, ν] b[ν, 3], a[4, ν] b[ν, 4]}}
```

Question:

1. Why expanding dummy from the beginning?
2. What about free indices, can we not to treat them as tables?

Indices Method 2: Abstract Indices

$a_{\mu\nu}b^{\nu\lambda}$ Let Mathematica keep abstract indices and learn the rules to do the algebra.

```
a[μ, ν] (b[ν, λ] + c[ν, λ]) // Expand
4]=
a[μ, ν] b[ν, λ] + a[μ, ν] c[ν, λ]
```

(Looks good here!)

We can even tell Mathematica upper/lower indices:

```
a[DN@μ, DN@ν] (b[UP@ν, UP@λ] + c[UP@ν, UP@λ]) // Expand
5]=
a[DN[μ], DN[ν]] b[UP[ν], UP[λ]] + a[DN[μ], DN[ν]] c[UP[ν], UP[λ]]
```

Can use Notation package to exactly display $a_{\mu\nu}b^{\nu\lambda} + a_{\mu\nu}c^{\nu\lambda}$

Indices Method 2: Abstract Indices

However, what if $s = a_\mu b^\mu$, we need s^2?

```
In[1]:= s = a[DN@μ] b[UP@μ]
Out[1]= a[DN[μ]] b[UP[μ]]

In[2]:= s^2
Out[2]= a[DN[μ]]^2 b[UP[μ]]^2
```

$$(a_\mu)^2 \, (b^\mu)^2$$

Things clearly go wrong. We know dummy should be unique.

But we did not tell Mathematica this rule.

Indices Method 2: Abstract Indices

How to generate unique dummies?

```
In[19]:=
    Unique[]
Out[19]=
    $3

In[20]:=
    Unique[]
Out[20]=
    $4

In[21]:=
    Unique[]
Out[21]=
    $5
```

Mathematica has provided a function.

Indices Method 2: Abstract Indices

```
In[3]:= s := With[{μ = Unique[]}, a[DN@μ] b[UP@μ]]

In[4]:= s * s

Out[4]= a[DN[$3]] a[DN[$4]] b[UP[$3]] b[UP[$4]]
```

$a_{\$3} a_{\$4} b^{\$3} b^{\$4}$ -- this is correct (though ugly).

Do not use s^2 -- still gives $(a_\mu)^2\ (b^\mu)^2$

One can replace \$n into other indices to:

- make it prettier
- enable cancellations between terms,

  e.g. ${a^{\$1}}_{\$1} - {a^{\$2}}_{\$2} = 0$ (How to ensure all cancellations?)

Indices Method 2: Abstract Indices

Tell free and dummy: e.g. $a^{\mu\nu\lambda}{}_{\nu} b_{\lambda\rho}$

```
tsr = a[UP@μ, UP@ν, UP@λ, DN@ν] b[DN@λ, DN@ρ]
a[UP[μ], UP[ν], UP[λ], DN[ν]] b[DN[λ], DN[ρ]]

Cases[tsr, (UP | DN)[μ_] :→ μ, Infinity] // Tally
{{μ, 1}, {ν, 2}, {λ, 2}, {ρ, 1}}

(* {list of free, list of dummy} *)
{Cases[%, {a_, 1} :→ a], Cases[%, {a_, 2} :→ a]}
{{μ, ρ}, {ν, λ}}
```

Indices Method 2: Abstract Indices

Decomposition of indices:

For example, 3+1 decomposition of metric. We need

$f^{\mu}{}_{\mu} = f^{0}{}_{0} + f^{i}{}_{i}$. How can this be done?

- Explicit indices: automatic done (summation is explicit)
- Abstract indices: Implement different types of indices

```
(* Work for one term after Expand.
  Need first to find out the dummy indices, and for each index do below. *)
f[UTot@μ, DTot@μ] /. {{UTot@μ → UP@i, DTot@μ → DN@i}, {UTot@μ → UE@0, DTot@μ → DE@0}} // Total

f[UE[0], DE[0]] + f[UP[i], DN[i]]
```

Indices Method 3: Mathematica build-in

TensorReduce

TensorReduce[*texpr*]
attempts to return a canonical form for the symbolic tensor expression *texpr*.

Details and Options

Examples (8)

Basic Examples (1)

Specify the properties of symbolic arrays:

In[1]:=
```
$Assumptions = {A ∈ Matrices[{3, 3}, Antisymmetric[{1, 2}]], S ∈ Arrays[{3, 3, 3}, Symmetric[{1, 2, 3}]]};
```

The trace of an antisymmetric matrix vanishes:

In[2]:=
```
TensorContract[A, {{1, 2}}] // TensorReduce
```
Out[2]=
```
0
```

The contraction of a symmetric and an antisymmetric pair vanishes:

In[3]:=
```
TensorContract[A⊗S, {{1, 3}, {2, 4}}] // TensorReduce
```
Out[3]=
```
0
```

Reorder tensor products lexicographically:

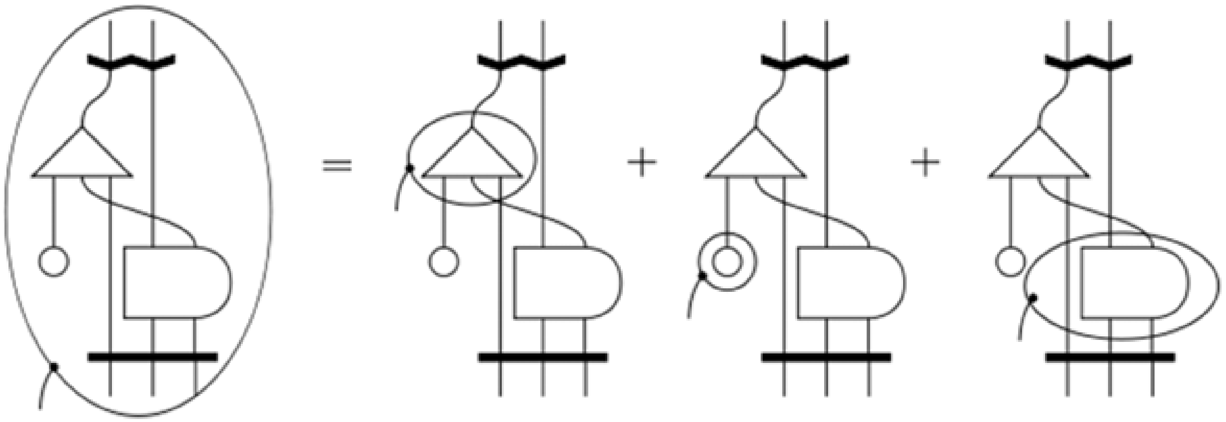
In[4]:=
```
S⊗A // TensorReduce
```
Out[4]=
```
TensorTranspose[A⊗S, {4, 5, 1, 2, 3}]
```

Indices Method 4: Graph Theory

Note: indices are not necessary for a tensor.

A tensor can be represented by a graph.

For example, Penrose's convention:



I am sure Mathematica can do it but did not implement it.

Packages: Overview

Explicit indices:

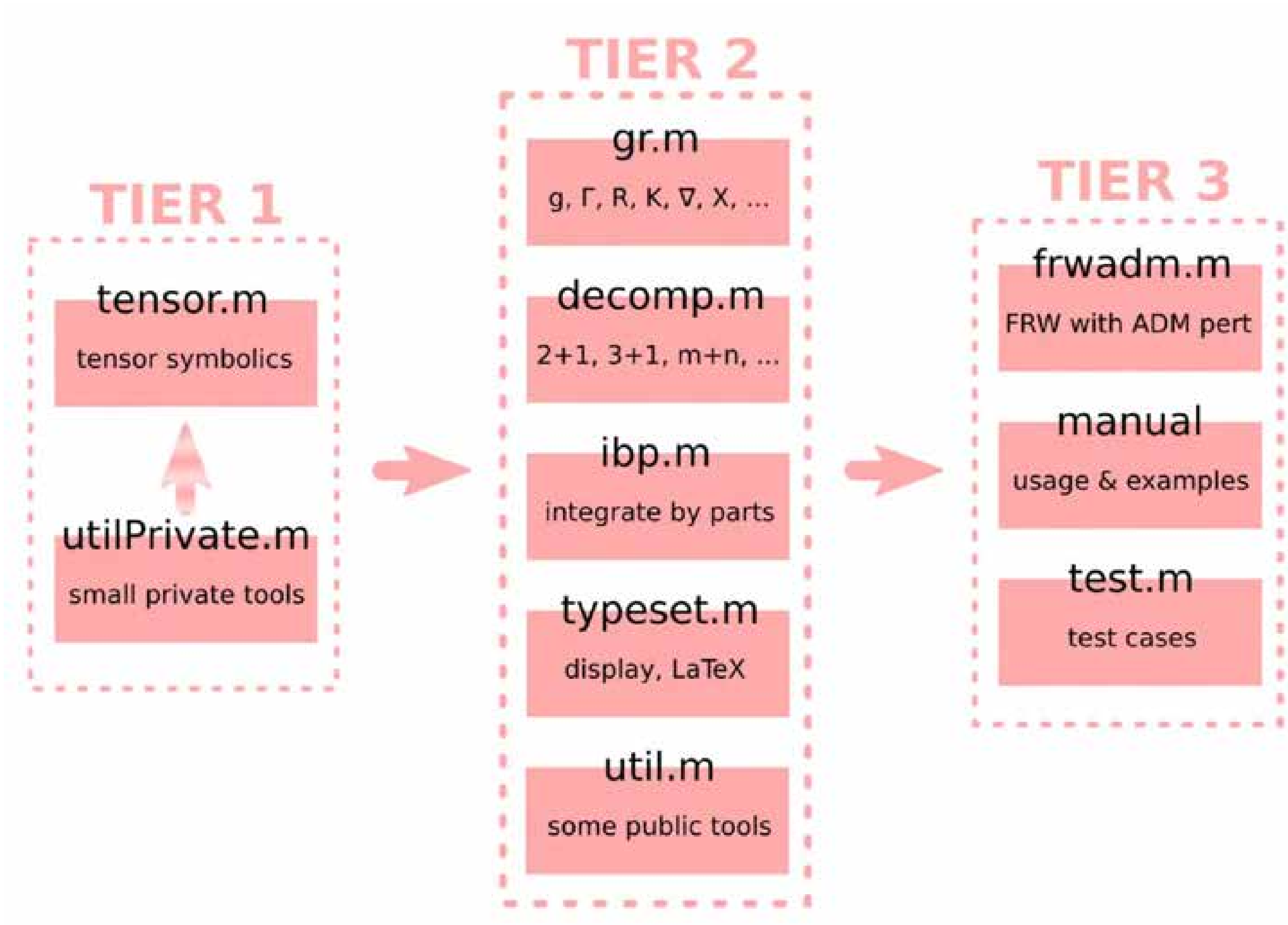
- Hartle's code (Mathematica)
- GRTensor (Maple)

Abstract indices:

- xAct (Mathematica)
- MathGR (Mathematica)
- Cadabra (C, LaTeX)

A long list of packages can be found at http://www.xact.es/links.html

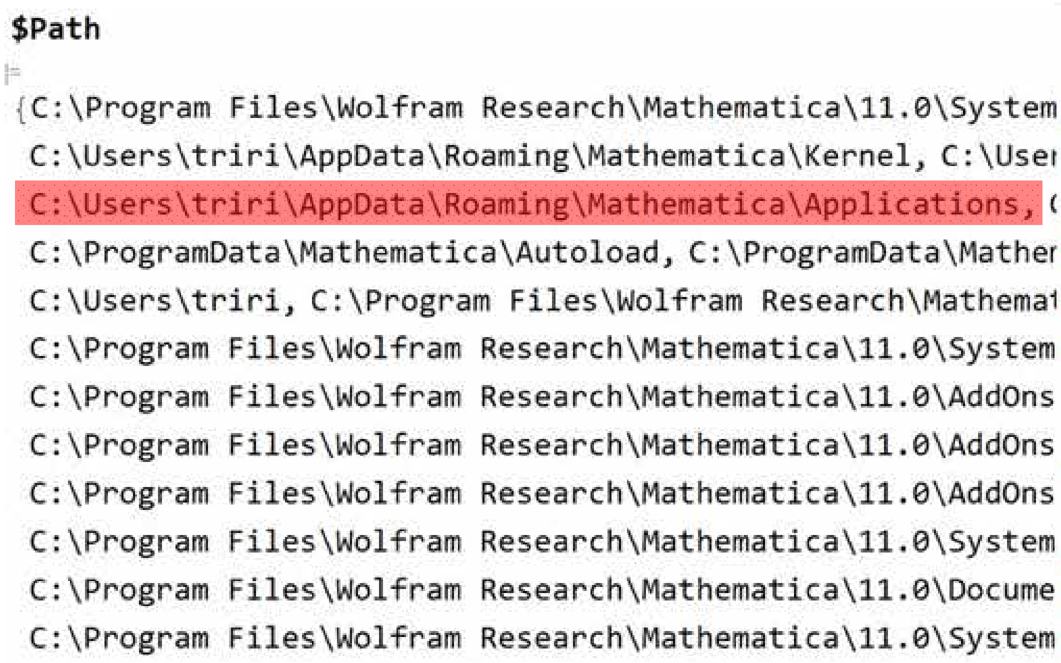
MathGR: Design & Structure

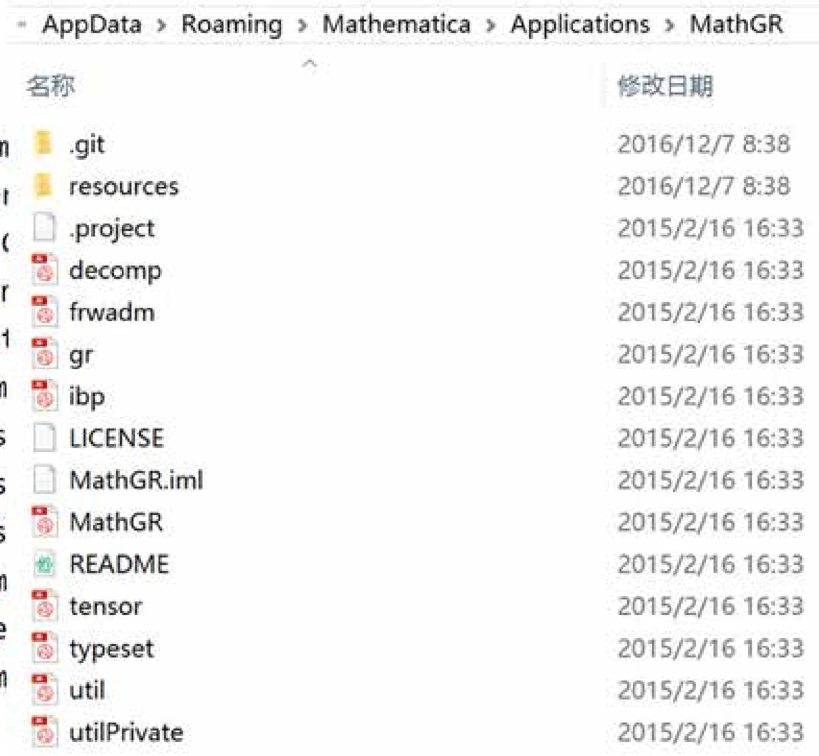


MathGR: Design & Structure

Installation:

- Get Code from https://github.com/tririver/MathGR/archive/v0.1.zip
- Put it in a path that Mathematica can find





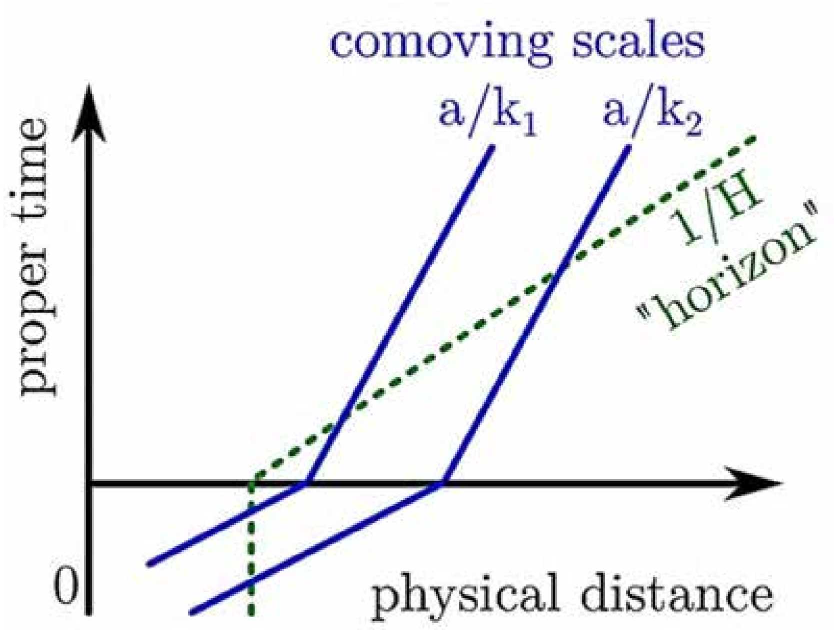
MathGR: Design & Structure

(Demo of 2nd order perturbations)

(Demo of gauge transformations)

(Demo of decomposition of dimensions)

From Primordial to CMB



At late times,

fluctuations return

to sub-Hubble scales.

CMB, LSS:

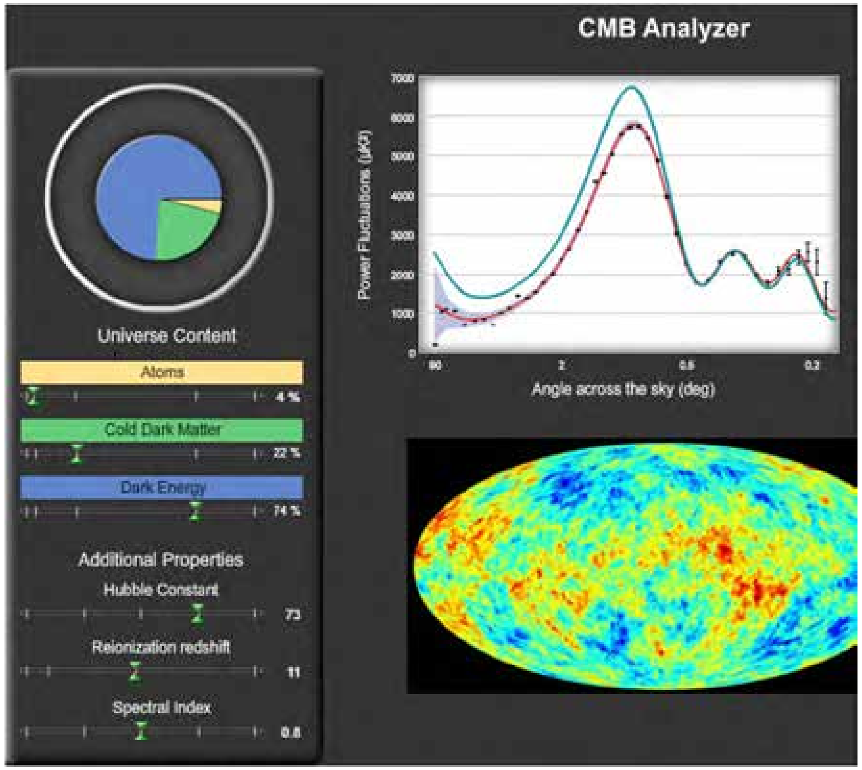
- Known fundamental physics
- Rich cosmological physics

Refer to Yacine's lectures for the theory of CMB

From Primordial to CMB

Can a “primordial” theorist use CMB data?

Yes, it’s simple. First see a toy example:



1. How primordial parameters “become” the spectrum and map?
2. Can computer tune those parameters by itself?

From Primordial to CMB

1.1 How primordial parameters “become” the CMB spectrum

That’s the whole theory of CMB. But there are codes:

- CMBFAST (no longer supported)
- CAMB (in Fortran, most widely used)
- CLASS (in C, has python interface, easier to read)
- CMBquick (Pitrou, in Mathematica)

Given your power spectrum, just run the code to see what your CMB spectrum look like.

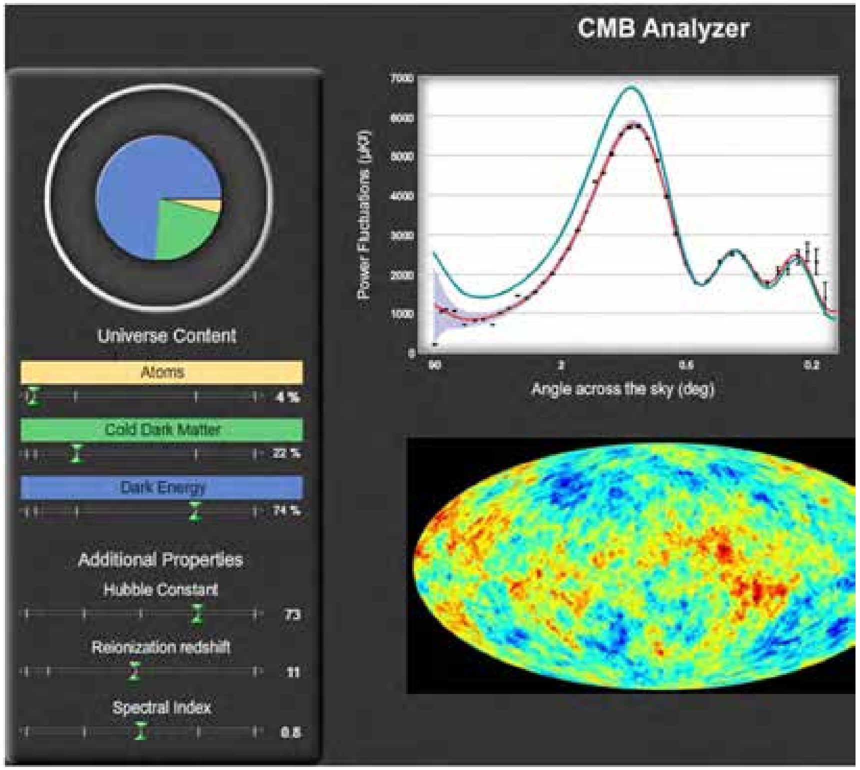
1.2 CMB map simulation: HEALPix and related tools.

From Primordial to CMB

2. Can computer tune those parameters by itself?

2.1 What is the standard of "result look like data"?

2.2 How to tune towards this goal?



From Primordial to CMB

2.1 What is the standard of “result look like data”? Likelihood.

Bayesian theorem: Given cosmological parameters $\boldsymbol{\theta}$,

$$P(\boldsymbol{\theta}|\text{data}) \propto P(\text{data}|\boldsymbol{\theta})\, P(\boldsymbol{\theta})$$

Prior. give flat if don’t know.

What we need. Given observational data, what’s the probability that your theory (with parameters $\boldsymbol{\theta}$) is right? Further: which parameters of your theory fits data better?

Likelihood. Given parameters (your theory), how well they fit data? (CAMB, CLASS)

From Primordial to CMB

2.2 How to tune towards this goal?

2.2.1 Provide best-fit values (too special)

2.2.2 Provide samples of $\boldsymbol{\theta}$ obeying $P(\boldsymbol{\theta}|\mathrm{data})$

Sampling: Markov Chain Monte Carlo

(see also, Nested Sampling, etc.)

Move a step → calculate likelihood ratio r:

$r > 1$ → accept move

$r < 1$ → accept move at probability r

From Primordial to CMB

Example: CosmoMC (with CAMB)

- Use GitHub version if one of the following:
  - you are good at Linux
  - Need latest version (e.g. latest experiments)
  - Need speed for serious calculation
- Use CosmoBox version if otherwise

From Primordial to CMB

Example: Monte Python (with CLASS)

Summary of This Lecture

Tools for cosmological perturbations

- Symbolic GR
  - Explicit indices
  - Abstract indices
- CMB codes to test your theory