Usage of *Mathematica* (M) Beyond a Calculator

Yi Wang, IPMU, June 4, 2013

M as a calculator :: Input quickly

Ctrl + shortcuts (tab to go through boxes):

$\frac{\square}{\square}$ (* / *), $\square^{\square}$ (* 6, i.e. ^ *), $\square_{\square}$ (* -, i.e. _ *), $\sqrt{\square}$ (* 2 *)

Esc + input + Esc

α (* a *), $\partial_{\square}\,\square$ (* dt *), $\int \square\, d\square$ (* intt *), $\int_{\square}^{\square} \square\, d\square$ (* dintt *)

M as a calculator :: Algebra 1

```
w (x + y + z) - w (x + y + z)
```

```
0
```

```
w (x + y + z) - w (x + y - z)
```

```
-w (x + y - z) + w (x + y + z)
```

```
w (x + y + z) - w (x + y - z) // Expand
```

```
2 w z
```

```
w (x + y + z) - w (x + y - z) // Simplify
```

```
2 w z
```

```
Simplify[Sqrt[x^2]]
```

$$\sqrt{x^2}$$

```
Simplify[Sqrt[x^2], x < 0]
```

```
-x
```

```
w / (x (y + z)) // Expand
```

$$\frac{w}{x (y + z)}$$

```
w / (x (y + z)) // ExpandAll
```

$$\frac{w}{x y + x z}$$

```
Simplify[x Gamma[x]]
```

```
x Gamma[x]
```

```
FullSimplify[x Gamma[x]]
```

```
Gamma[1 + x]
```

M as a calculator :: Algebra 2

```
ds2 = -(1 - rs/r) dt^2 + (1 + rs/r) dr^2 // Expand
```

$$dr^2 - dt^2 + \frac{dr^2\, rs}{r} + \frac{dt^2\, rs}{r}$$

```
Collect[ds2, {dt, dr}, Expand]
```

$$dt^2 \left(-1 + \frac{rs}{r}\right) + dr^2 \left(1 + \frac{rs}{r}\right)$$

Use case of Collect:
(a) Get good looking result
(b) Divide long expressions and ease simplification

```
ClearAll[ds2]
```

M as a calculator :: Algebra 3

Some other useful tricks

```
Re[E^(x+I y)]
```

```
Re[ℯ^(x+i y)]
```

```
FullSimplify[Re[E^(x+I y)], x > 0 && y > 0]
```

```
Re[ℯ^(x+i y)]
```

```
ComplexExpand[Re[E^(x+I y)]]
```

```
ℯ^x Cos[y]
```

```
ComplexExpand[Re[E^(x+I y+c)], {c}]
```

```
ℯ^(x+Re[c]) Cos[y + Im[c]]
```

```
E^(x+I y) // ExpToTrig
```

```
Cosh[x + i y] + Sinh[x + i y]
```

```
Cosh[x + i y] + Sinh[x + i y] // TrigToExp
```

```
ℯ^(x+i y)
```

M as a calculator :: Calculus

```
∂x Sin[x]
```

```
Cos[x]
```

```
∫ Cos[x] dx
```

```
Sin[x]
```

```
Sum[1/n^x, {n, 1, ∞}]
```

```
Zeta[x]
```

```
Series[BesselJ[ν, z], {z, 0, 2}] // FullSimplify
```

$$z^{\nu}\left(\frac{2^{-\nu}}{\text{Gamma}[1+\nu]}-\frac{2^{-2-\nu}\,z^2}{\text{Gamma}[2+\nu]}+O[z]^3\right)$$

```
Series[HankelH1[ν, z], {z, ∞, 2}] // FullSimplify
```

$$e^{i z}\left(\frac{(1-i)\,e^{-\frac{1}{2}i\pi\nu}\sqrt{\frac{1}{z}}}{\sqrt{\pi}}+\frac{1}{\sqrt{\pi}}\left(\frac{1}{8}+\frac{i}{8}\right)e^{-\frac{1}{2}i\pi\nu}\left(-1+4\nu^2\right)\left(\frac{1}{z}\right)^{3/2}+O\left[\frac{1}{z}\right]^{5/2}\right)$$

```
DSolve[3 H[t] ∂t ϕ[t] + m^2 ϕ[t] == 0, ϕ[t], t]
```

$$\left\{\left\{\phi[t]\to e^{\int_1^t -\frac{m^2}{3H[K[1]]}dK[1]}\,C[1]\right\}\right\}$$

```
DSolve[∂t ϕ[t, x] + ∂x ∂x ∂x ϕ[t, x] - 6 ϕ[t, x] ∂x ϕ[t, x] == 0, ϕ[t, x], {t, x}]
```

DSolve::nlpde : Solution requested to nonlinear partial differential equation. Trying to build a special solution. ≫

$$\left\{\left\{\phi[t,x]\to\frac{1}{6\,C[2]}\left(C[1]-8\,C[2]^3+12\,C[2]^3\,\text{Tanh}[t\,C[1]+x\,C[2]+C[3]]^2\right)\right\}\right\}$$

M as a calculator :: Numerics

```
SetAccuracy[1 / 2 - 1 / 4, 100]
```

```
0.250000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
 000000000000000000000
```

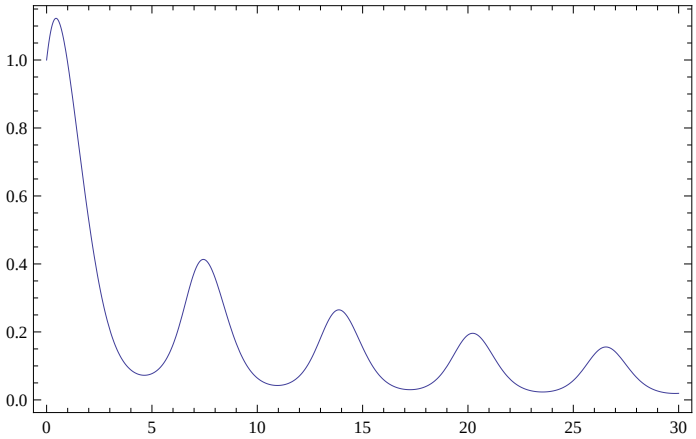
```
NIntegrate[Sin[x] / x, {x, 0, π}, PrecisionGoal → 100, WorkingPrecision → 100]
```

```
1.851937051982466170361053370157991363345809728981154909804783781876981890166348358532710336502954758
```

```
s = NDSolve[{y'[x] == y[x] Cos[x + y[x]], y[0] == 1}, y, {x, 0, 30}]
```

```
{{y → InterpolatingFunction[{{0., 30.}}, <>]}}
```

```
Plot[Evaluate[y[x] /. s], {x, 0, 30}, PlotRange → All, Frame → True]
```



Edit the plot using Ctrl+d, use "save graphic as" to export

See also Plot3D, ParametricPlot, DensityPlot

```
ClearAll[s]
```

M as a calculator :: Notation

```
Needs["Notation`"]
```

```
Notation[ f_ ⟺ f_[x, y, z, t] ]
```

```
{δϕ[x, y, z, t], D[f[x, y, z, t], t]}
```

$$\left\{\underline{\delta\phi}, \underline{\mathbf{f}^{(0,0,0,1)}}\right\}$$

```
{δϕ, f^(0,0,0,1)} // InputForm
```

```
{δϕ[x, y, z, t], Derivative[0, 0, 0, 1][f][x, y, z, t]}
```

```
RemoveNotation[ f_ ⟺ f_[x, y, z, t] ]
```

```
Notation[ f_a_b_ ⟺ f_[DN[a_], DN[b_]] ]
```

```
Notation[ f_^a_b_ ⟺ f_[UP[a_], UP[b_]] ]
```

```
Notation[ f_ a_ _b_ ⟺ f_[UP[a_], DN[b_]] ]
```

```
g[DN[μ], DN[ν]] x[UP[μ], UP[ν]]
```

$$\mathbf{g}_{\mu\nu}\, \mathbf{x}^{\mu\nu}$$

```
g_μν // InputForm
g[DN[μ], DN[ν]]
```

Term rewriting :: Set vs SetDelayed

First some intuition:

```
a = Random[] (* i.e. Set[a, Random[]] *)
```

```
0.419947
```

```
{a, a, a}
```

```
{0.419947, 0.419947, 0.419947}
```

```
b := Random[](* i.e. SetDelayed[a, Random[]] *)
```

```
{b, b, b}
```

```
{0.635014, 0.717431, 0.479858}
```

How a and b are calculated:

```
Trace[a]
```

```
{a, 0.419947}
```

```
Trace[b]
```

```
{b, Random[], 0.221315}
```

```
Attributes[{Set, SetDelayed}]
```

```
{{HoldFirst, Protected, SequenceHold}, {HoldAll, Protected, SequenceHold}}
```

```
ClearAll[a, b]
```

Term rewriting :: Rule vs RuleDelayed

```
{c, c, c} /. c → Random[]
```

```
{0.336622, 0.336622, 0.336622}
```

```
{c, c, c} /. c :→ Random[]
```

```
{0.130798, 0.206383, 0.420582}
```

```
Trace[c /. c → Random[]]
```

```
{{{Random[], 0.389959}, c → 0.389959, c → 0.389959}, c /. c → 0.389959, 0.389959}
```

```
Trace[c /. c :→ Random[]]
```

```
{{c :→ Random[], c :→ Random[]}, c /. c :→ Random[], Random[], 0.912726}
```

Set and SetDelayed : global rules
Rule and RuleDelayed: local rules to be applied using /. (ReplaceAll)

ReplaceAll (/.) vs ReplaceRepeated (//.)

```
{a'[t], a''[t]} /. {a'[t] → H[t] a[t], a''[t] → ∂t (H[t] a[t]), H'[t] → -ϵ[t] H[t]^2}
```

```
{a[t] H[t], H[t] a'[t] + a[t] H'[t]}
```

```
{a'[t], a''[t]} //. {a'[t] → H[t] a[t], a''[t] → ∂t (H[t] a[t]), H'[t] → -ϵ[t] H[t]^2}
```

```
{a[t] H[t], a[t] H[t]^2 - a[t] H[t]^2 ϵ[t]}
```

Term rewriting :: Patterns

Patterns makes the whole thing (=, :=, ->, :>) much stronger!

A pattern matches a class of exprs instead of a special expr.
_ is a pattern that matches every single expr. x_ (abbrev of x:_) gives name x to _

```
ClearAll[f, d]
f[x_] := myFunction[x]
{f[1], f[a + b c d], f["string"], f[a, b]}
```

```
{myFunction[1], myFunction[a + b c d], myFunction[string], f[a, b]}
```

```
d[x_ + y_] := d[x] + d[y];
d[x_ * y_] := x * d[y] + d[x] * y;
d[a + b * c]
```

```
d[a] + c d[b] + b d[c]
```

```
d1[a + b * c] //. {d1[x_ + y_] :> d1[x] + d1[y], d1[x_ * y_] :> x * d1[y] + d1[x] * y}
```

```
d1[a] + c d1[b] + b d1[c]
```

Similarly __ (2 _'s together) matches 1 or more stuffs, ___ (3 _'s) matches 0 or more stuffs

```
Fold[d, f, {a, b, c}]
```

```
d[d[d[f, a], b], c]
```

```
d[f, a, b, c]
```

```
d[f, a, b, c]
```

```
d[f_][a__] := Fold[d, f, {a}]
```

```
d[f][a, b, c]
```

```
d[d[d[f, a], b], c]
```

```
d[f][a, b, c, x, y, z]
```

```
d[d[d[d[d[f, a], b], c], x], y], z]
```

```
ClearAll[f, d]
```

Term rewriting :: Test / collect matches

Find expressions that match patterns: MatchQ, FreeQ, Cases, ...

MatchQ[*expr*, *pattern*]: test if *pattern* matches *expr*

```
{MatchQ[x^2 + 2 x + 1, a_^2], MatchQ[x^2, a_^2]}
```

```
{False, True}
```

FreeQ[*expr*, *pattern*]: test if *expr* is free of *pattern*

```
{FreeQ[x^2 + 2 x + 1, a_^2], FreeQ[2 x + 1, a_^2]}
```

```
{False, True}
```

```
ClearAll[δ]
δ /: δ[a_, b_] f_ := Piecewise[{
    {f /. b → a, ! FreeQ[f, b]},
    {f /. a → b, ! FreeQ[f, a]}
  }, HoldForm[δ][a, b] f]
```

```
{h_ab δ_ac, δ_ab δ_ac, h_mn δ_ac}
```

```
{h_cb, δ_bc, h_mn δ_ac}
```

Cases[*expr*, *pattern*, *level*]: find all sub-expressions in *expr*, that match *pattern* at given *level*.

```
g_μν x^μρ x^νλ // InputForm
```

```
g[DN[μ], DN[ν]]*x[UP[μ], UP[ρ]]*x[UP[ν], UP[λ]]
```

```
Cases[g_μν x^μρ x^νλ, UP[_] | DN[_], Infinity]
```

```
{DN[μ], DN[ν], UP[μ], UP[ρ], UP[ν], UP[λ]}
```

```
idx = Cases[g_μν x^μρ x^νλ, (UP | DN)[i_] :> i, Infinity]
```

```
{μ, ν, μ, ρ, ν, λ}
```

```
Tally[idx]
```

```
{{μ, 2}, {ν, 2}, {ρ, 1}, {λ, 1}}
```

```
dummy = Cases[Tally[idx], {i_, 2} :> i]
```

```
{μ, ν}
```

```
free = Cases[Tally[idx], {i_, 1} :> i]
```

```
{ρ, λ}
```

```
ClearAll[idx, dummy, free]
```

Term rewriting :: Pattern test

For procedural programming, if...then... is important (to do anything “clever”)
For term rewriting, *pattern test* plays the role of if...then...

```
ClearAll[f0]
f0[a_ /; a > 0] := Sqrt[a]
```

```
{f0[2], f0[-2], f0["string"]}
```

$$\left\{\sqrt{2}\text{, f0[-2], f0[string]}\right\}$$

```
ClearAll[δ]
δ /: δ[a_, b_] f_ /; ! FreeQ[f, b] := f /. b → a
δ /: δ[a_, b_] f_ /; ! FreeQ[f, a] := f /. a → b
```

```
{h_ab δ_ac, δ_ab δ_ac, h_mn δ_ac}
```

$$\{h_{cb},\ \delta_{bc},\ h_{mn}\ \delta_{ac}\}$$

```
ClearAll[f]
f[n_ /; n > 1] := n f[n - 1]
f[1] := 1
```

```
f[5]
```

```
120
```

For single argument function, there is alterform of pattern test: ?

```
Cases[{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20}, a_?PrimeQ]
```

```
{2, 3, 5, 7, 11, 13, 17, 19}
```

```
Select[{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20}, PrimeQ]
```

```
{2, 3, 5, 7, 11, 13, 17, 19}
```

Term rewriting :: How M works (1)

The *Mathematica* way:
abs[x_ /;x>=0] := x
abs[x_ /;x<0] := -x

Find an applicable rule, replace the expression, until no more rules applicable.

This (term rewriting paradigm) is very different from, e.g.

The *C* way:

```
int abs(int x){
   x>0 ? x : -x;
}
```

We can either write C-style M code like
abs[x_] := If[x>=0, x, -x]
But again, this function is understood by a rule in M.

Functional programming :: The concept

Function of function of ... of function can be written cleanly

y = Sqrt @ Abs @ Sin @ x

instead of tmp1=Sin[x]; tmp2=Abs[tmp1]; y=Sqrt[tmp2]

Advantage:

- Short and no distant action, thus easier to understand / test
- No temp vars, thus fast (M is interpret lang, cannot optimize temp vars into CPU registers)
- No side effect, thus different copies can be parallelized

Assumption:

(1) every function takes one argument

(2) every function is defined.

However,

(1) the grammar maybe complicated, e.g.

y1 = Sqrt @ (1 + Abs @ Sin @ x)

(2) one may need to define function separately, e.g.

myFunc[x_] := x^2 - 2*x + 1

y2 = Sqrt @ myFunc @ Abs @ Sin @ x

Both can be overcame by "λ-calculus" (pure function): λx.(how to calculate x)

in M: Function[x, (how to calculate x)], or simply (how to calculate #)&

e.g. Sin[#]& @ x == Sin[x], Plus[1 + #]& @ x == 1+x . Above example:

y1 = Sqrt @ Plus[1 + #]& @ Abs @ Sin @ x

y2 = Sqrt @ (#^2 - 2*# + 1 &) @ Abs @ Sin @ x

Then the functional workflow is preserved (and important Math applications!).

```
(Collect[#, {dr, dt}, Expand] &) @ (# /. {z → 1 / r} &) @ (dr^2 - dt^2 + dr^2 rs z + dt^2 rs z)
```

$$dt^2 \left(-1 + \frac{rs}{r}\right) + dr^2 \left(1 + \frac{rs}{r}\right)$$

```
(dr^2 - dt^2 + dr^2 rs z + dt^2 rs z) // (# /. {z → 1 / r} &) // (Collect[#, {dr, dt}, Expand] &)
```

$$dt^2 \left(-1 + \frac{rs}{r}\right) + dr^2 \left(1 + \frac{rs}{r}\right)$$

Generalization to multi-variable functions:

```
Sort[{a_2, c_1, b_3}]
```

$$\{a_2, b_3, c_1\}$$

```
Sort[{a_2, c_1, b_3}, #1[[2]] < #2[[2]] &]
```

$$\{c_1, a_2, b_3\}$$

Who said code and data are separated (Von Neumann)?

```
SortBy[{a2, c1, b3}, Last]
```

```
{c1, a2, b3}
```

Functional programming :: pre- post- and in-fix

```
f[x] === f@x
```

```
True
```

```
f[x] === (x // f)
```

```
True
```

```
f[x, y] === x~f~y
```

```
True
```

```
a~Plus~b
```

```
a + b
```

```
"This is "~StringJoin~"a string"
```

```
This is a string
```

```
"This is " <> "a string"
```

```
This is a string
```

```
{a, b, c}~Join~{x, y, z}
```

```
{a, b, c, x, y, z}
```

Functional programming :: List processing

Importance of list {a,b,c,...} in functional programming:

- unify code and data
- ready to parallelize (ParallelMap, ParallelTable, ...)
- way of thinking and coding: construct a list and apply options on it

Frequently used functions on list:

- Creating list: Table, Range
- Take elements from list: Take[list, elem], i.e. list[[elem]]
- Modifications: Append, Prepend, list[[elem]]=..., Flatten
- Selections: Select, Cases, ...
- Operations: Transpose, Reverse, Sort, Apply, Map, ...

```
testList = Range[5]
```

```
{1, 2, 3, 4, 5}
```

```
testList[[2]] (* 2nd element *)
```

```
2
```

```
testList[[2 ;;]] (* from 2nd to last *)
```

```
{2, 3, 4, 5}
```

```
testList[[2 ;; 4]](* from 2nd to 4th *)
```

```
{2, 3, 4}
```

```
testList[[-2 ;;]] (* last 2 elements *)
```

```
{4, 5}
```

```
Apply[f, testList]
```

```
f[1, 2, 3, 4, 5]
```

```
f @@ testList
```

```
f[1, 2, 3, 4, 5]
```

```
Map[f, testList]
```

```
{f[1], f[2], f[3], f[4], f[5]}
```

```
f /@ testList
```

```
{f[1], f[2], f[3], f[4], f[5]}
```

```
ParallelMap[f, testList]
```

```
{f[1], f[2], f[3], f[4], f[5]}
```

c.f. Unix philosophy: Do one thing and do it well.
usage of pipe ≃ functional programming

```
        cat file | grep “key word” > line_with_key_word.txt
```

xargs ≃ Map

```
        echo file1 file2 file3 file4 | xargs touch
```

Functional programming :: How M works (2)

The data structure of M:

```
{"a", "b", "c"}[[1]]
```

```
a
```

```
{"a", "b", "c"}[[0]]
```

```
List
```

```
ClearAll[a, b, c]
{(a + b + c)[[0]], (a + b + c)[[1]]}
```

```
{Plus, a}
```

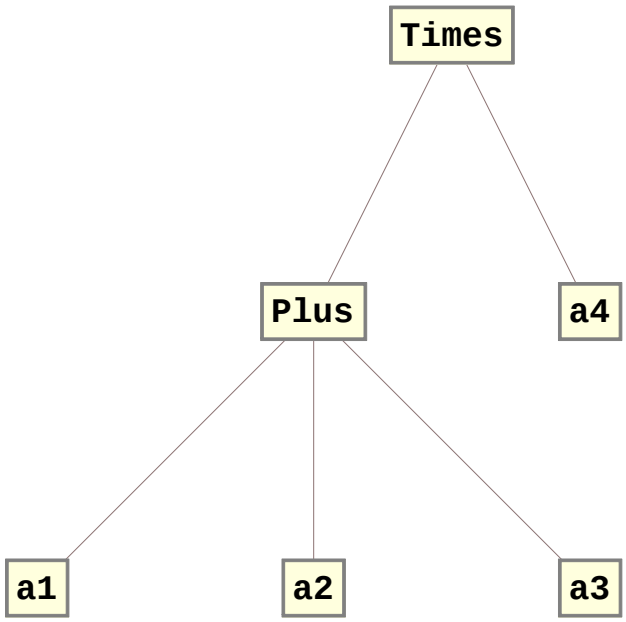
```
List @@ (a + b + c)
```

```
{a, b, c}
```

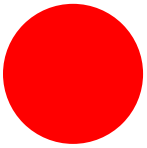
```
f /@ (a + b + c)
```

```
f[a] + f[b] + f[c]
```

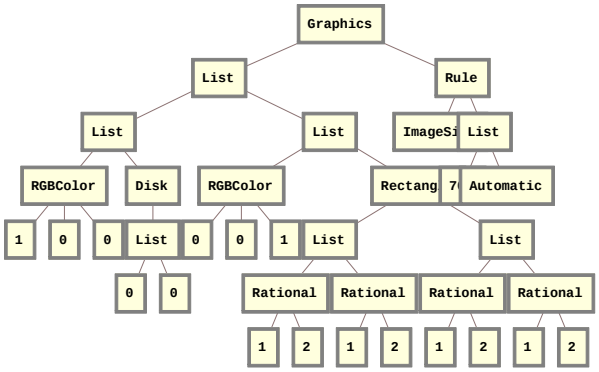
```
(a1 + a2 + a3) * a4 // TreeForm
```



Thus M is a good looking analogue of LISP (with subtle difference)
The LISP way: (* (+ 1 2 3) 4)



// **TreeForm**


Graphics
List
Rule
List
List
ImageSi
List
RGBColor
Disk
RGBColor
Rectang
7(
Automatic
1
0
0
List
0
0
1
List
List
0
0
Rational
Rational
Rational
Rational
1
2
1
2
1
2
1
2

Package management :: Notebook (.nb) vs Package (.m)

Package: a text file with extension .m and content like:

```
BeginPackage[“MathGR`look`”, {“MathGR`tensor`”}]
TensorFormatList::usage = “A list of formatted tensors with superscript- and subscript- idx style”
Begin[“`Private`”]
        ......
End[]
EndPackage[]
```

\	Notebook	Package
Ease of usage	WYSIWYG	Import and run
Code reusabiity	copy / paste	full
Namespace	by hand	good support
Editor / IDE	FrontEnd	Many choices

Who needs packages?

- Have reusable code
- Want to build good and improving tools (otherwise just copy / paste code)

Note: one can auto save code cells of Notebook into a package, by running
SetOptions[EvaluationNotebook[],AutoGeneratedPackage→Automatic];

Package management :: Local vars

```
f0[x_] := x
```

```
f0[-1]
```

```
-1
```

```
f0[2]
```

$\sqrt{2}$

How to avoid this problem?

- Short code: Just remember don't to it.
- Long code: Increasingly hard
- Use other people's package: we never know...

Naive solution: prefix (package name, module name, variable)

e.g. MathGR_gr_$R_{\mu\nu}$=...

Long variable names -> long code -> hard to read / maintain

Modern solution: local vars and name space

Local vars for a function is familiar (e.g. in C)

E.g. f[x_]:=Module[{metric},...] (* lexical scoping, *use this* for typical cases *)

E.g. f[x_]:=Block[{metric},...] (* dynamic scoping *)

Package management :: Namespace

Local vars are not enough:
    What if vars / functions want to be global in package but not exposed to users?

BeginPackage[“MathGR`look`”, {“MathGR`tensor`”}]
TensorFormatList::usage = “A list of formatted tensors with superscript- and subscript- idx style”
Begin[“`Private`”]
    aLocalFunc[x_]:=...
End[]
EndPackage[]

TensorFormatList can be used directly from outside (who used Get[“MathGR/look.m”])
aLocalFunc[x] is not accessible
    If you really want to call aLocalFunc from outside package, use MathGR`look`Private`aLocal-Func[x]

Underlying mechanism: Context.

```
Remove[myF, MyContext`myF]
```

```
Remove::remal : Symbol Removed[myF] already removed. ≫
```

```
Begin["MyContext`"]
```

```
MyContext`
```

```
Context[]
```

```
MyContext`
```

```
myF[x_] := x
```

```
myF[3.14]
```

```
3.14
```

```
End[]
```

```
MyContext`
```

```
Context[]
```

```
Global`
```

```
myF[3.14]
```

```
myF[3.14]
```

```
MyContext`myF[3.14]
```

```
3.14
```

Package management :: Eclipse

Better Editor

Organized windows

Simple unit / integration tests

Profiler

Example: a GR package

Motivation: A *simple* package with

- Tensor with explicit indices (dummy for sum)
- Tensor simplification with symmetries
- Support abstract and explicit indices
- Support decompose from abstract to explicit indices
- Integration by parts
- Cosmic perturbations
- Good-looking display

Keep it simple:

- now < 400 LOC ,<200 pure LOC (M line can be long though)
  c.f. xAct: xTensor.m alone >8000 LOC, whole package >20000 LOC
  Ricci.m 7580 LOC
- Modular, easy to develop / understand / fork and hack
- Make use of latest technology (M9.0 for full functionality), fast

Get it: Send email to *tririverwangyi@gmail.com* , with title *MathGR acquire package*

- The package is still in pre-release stage (may not work well)!

Example: a GR package :: Architecture

tensor.m : Core for tensor manipulation

- Get free / dummy idx
- Partial derivative
- Define tensor symmetry
- Simplify tensor into unique form
- Decompose idx
- Delta symbol

gr.m : metric, contract with metric, CovD, connection & curvature tensors

ibp.m : bring expr into total derivative with “minimal” rest part

util.m : utilities for Fourier transformation, series expansion, solving eqs.

utilPrivate.m : private tools used for package internal (no API to end users)

look.m : display the tensors in conventional upper-lower indices

frwadm.m : defines FRW background metric with ADM perturbations

Example: a GR package :: Sample calculation

```
Quit[]
```

```
Get["MathGR/frwadm.m"];
Get["MathGR/look.m"];
```

```
DeclareSym[zz, {UP, UP, DN, DN}, Symmetric[{1, 2}]];
DeclareSym[zz, {UP, UP, DN, DN}, Antisymmetric[{3, 4}]];
zz[UP@"i", UP@"j", DN@"i", DN@"j"]
```

$$zz^{ij}{}_{ij}$$

```
zz[UP@"i", UP@"j", DN@"i", DN@"j"] // Simp
```

```
0
```

```
WithMetric[g, R[] // Simp]
```

$$-\left(\partial_{d}\,\partial_{c}\,g_{ab}\right)\left(g^{ab}\right)\left(g^{cd}\right)+\left(\partial_{d}\,\partial_{b}\,g_{ac}\right)\left(g^{ab}\right)\left(g^{cd}\right)+\frac{3}{4}\left(\partial_{e}\,g_{ac}\right)\left(\partial_{f}\,g_{bd}\right)\left(g^{ab}\right)\left(g^{cd}\right)\left(g^{ef}\right)-\frac{1}{2}\left(\partial_{f}\,g_{ac}\right)\left(\partial_{d}\,g_{be}\right)\left(g^{ab}\right)\left(g^{cd}\right)\left(g^{ef}\right)-\left(\partial_{d}\,g_{ac}\right)\left(\partial_{f}\,g_{be}\right)\left(g^{ab}\right)\left(g^{cd}\right)\left(g^{ef}\right)-\frac{1}{4}\left(\partial_{e}\,g_{ab}\right)\left(\partial_{f}\,g_{cd}\right)\left(g^{ab}\right)\left(g^{cd}\right)\left(g^{ef}\right)+\left(\partial_{d}\,g_{ab}\right)\left(\partial_{f}\,g_{ce}\right)\left(g^{ab}\right)\left(g^{cd}\right)\left(g^{ef}\right)$$

```
AbsoluteTiming@WithMetric[g,
  CovD[R[-a, -b, -c, -d], DN["e"]] + CovD[R[-a, -b, -e, -d], DN["c"]] + CovD[R[-a, -b, -c, -e], DN["d"]] // Simp]
```

```
{1.545873, 0}
```

The above test takes xAct 25 secs, with pre-compiled C-library.

Example: a GR package :: Sample calculation

```
PdHold[__] := 0; (* don't care total derivatives *)
ϕ = ϕ0;
Pd[ϕ0, _DN] := 0;
s012 = Sqrtg (RADM[] / 2 + DecompG2H[X[ϕ]] - V[ϕ] ) // SS[2]
```

$$-3\,a^3\,H^2+\frac{1}{2}\,a^3\,\dot{\phi 0}^2-a^3\,V[\phi 0]+\frac{1}{2}\,Eps\left(6\,a^3\,H^2\,\alpha-18\,a^3\,H^2\,\zeta-12\,a^3\,H\,\dot{\zeta}-a^3\,\alpha\,\dot{\phi 0}^2+3\,a^3\,\zeta\,\dot{\phi 0}^2+4\,a\,H\left(\partial^2\,\beta\right)-4\,a\left(\partial^2\,\zeta\right)-2\,a^3\,\alpha\,V[\phi 0]-6\,a^3\,\zeta\,V[\phi 0]\right)+\frac{1}{4\,a}\,Eps^2\left(-12\,a^4\,H^2\,\alpha^2+36\,a^4\,H^2\,\alpha\,\zeta-54\,a^4\,H^2\,\zeta^2+24\,a^4\,H\,\alpha\,\dot{\zeta}-72\,a^4\,H\,\zeta\,\dot{\zeta}-12\,a^4\,\dot{\zeta}^2+8\,a^2\,H\left(b_a\right)\left(\partial_a\,\zeta\right)+8\,a^2\,H\left(\partial_a\,\beta\right)\left(\partial_a\,\zeta\right)-4\,a^2\left(\partial_a\,\zeta\right)^2+2\,a^4\,\alpha^2\,\dot{\phi 0}^2-6\,a^4\,\alpha\,\zeta\,\dot{\phi 0}^2+9\,a^4\,\zeta^2\,\dot{\phi 0}^2+\left(\partial_b\,b_a\right)^2+\left(\partial_b\,b_a\right)\left(\partial_a\,b_b\right)-8\,a^2\,H\,\alpha\left(\partial^2\,\beta\right)+8\,a^2\,H\,\zeta\left(\partial^2\,\beta\right)+8\,a^2\,\dot{\zeta}\left(\partial^2\,\beta\right)+4\left(\partial_b\,b_a\right)\left(\partial_b\,\partial_a\,\beta\right)+2\left(\partial_b\,\partial_a\,\beta\right)^2-2\left(\partial^2\,\beta\right)\left(\partial^2\,\beta\right)-8\,a^2\,\alpha\left(\partial^2\,\zeta\right)-8\,a^2\,\zeta\left(\partial^2\,\zeta\right)-12\,a^4\,\alpha\,\zeta\,V[\phi 0]-18\,a^4\,\zeta^2\,V[\phi 0]\right)$$

```
s1 = s012 // OO[1]
```

$$3\,a^3\,H^2\,\alpha-9\,a^3\,H^2\,\zeta-6\,a^3\,H\,\dot{\zeta}-\frac{1}{2}\,a^3\,\alpha\,\dot{\phi 0}^2+\frac{3}{2}\,a^3\,\zeta\,\dot{\phi 0}^2+2\,a\,H\left(\partial^2\,\beta\right)-2\,a\left(\partial^2\,\zeta\right)-a^3\,\alpha\,V[\phi 0]-3\,a^3\,\zeta\,V[\phi 0]$$

```
SimpHook = SimpHook ~Union~ (SolveExpr[
      {D[s1, α] == 0, D[Ibp[s1, IbpVar[ζ]], ζ] == 0}, {V[ϕ0], Pd[ϕ0, DE@0]^2}][[1]])
```

$$\left\{\text{Dim}\to 3,\ \dot{a}\to a\,H,\ \dot{H}\to -H^2\,\epsilon,\ \dot{\epsilon}\to H\,\epsilon\,\eta,\ \dot{\phi 0}^2\to 2\,H^2\,\epsilon,\ V[\phi 0]\to 3\,H^2-H^2\,\epsilon\right\}$$

```
s2 = s012 // OO[2] // Fourier2
```

$$-3\,a^3\,H^2\,\alpha^2+2\,a\,H\,k^2\,\alpha\,\beta+a^3\,H^2\,\alpha^2\,\epsilon+2\,a\,k^2\,\alpha\,\zeta-27\,a^3\,H^2\,\zeta^2+a\,k^2\,\zeta^2+9\,a^3\,H^2\,\epsilon\,\zeta^2+\frac{k^2\left(b_a\right)^2}{4\,a}+6\,a^3\,H\,\alpha\,\dot{\zeta}-2\,a\,k^2\,\beta\,\dot{\zeta}-18\,a^3\,H\,\zeta\,\dot{\zeta}-3\,a^3\,\dot{\zeta}^2$$

```
solCons =
 Solve[{D[s2, α] == 0,  D[s2, β] == 0, D[s2, b[DN@"a"]] == 0}, {α, β, b[DN@"a"]}][[1]]
```

$$\left\{\alpha\to\frac{\dot{\zeta}}{H},\ \beta\to-\frac{k^2\,\zeta+a^2\,H\,\epsilon\,\dot{\zeta}}{H\,k^2},\ b_a\to 0\right\}$$

```
s2Solved = s2 /. solCons // Ibp[#, IbpStd2] &
```

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

M as a calculator

Term rewriting

Functional programming

Package management

Example: a GR package

SayTo[x_?CameQ]:= “Thanks, ”<>x<>“!”

SayTo /@{...}

Further reading :

- Press F1 in *Mathematica*
- *Mathematica programming: an advanced introduction*, by Leonid Shifrin
- *Power programming with Mathematica: the kernel*, by David B. Wagner

Get MathGR : Send email to *tririverwangyi@gmail.com* , with title *MathGR acquire package*

- The package is still in pre-release stage (may not work well)!