

Computer Aided Mexcio Tests

Kranc

Kranc assembles numerical code

S. Husa, I. Hinder, C. Lechner

AEI Golm

University of Southampton

Mexico, December 2003

Idea

Idea

$$G_{ab} = 0$$

Idea

$$G_{ab} = 0$$



Kranc

Idea

$$G_{ab} = 0$$



Kranc



3+1 decomposition
analyze equations

Idea

$$G_{ab} = 0$$



Kranc



3+1 decomposition
analyze equations



Cactus Thorns

Kranc consists of

Mathematica scripts and notebook templates

Tools for tensor analysis e.g. 3+1 decomposition, analyzing the principal part etc.

tools and templates use [MathTensor](#),

MathTensor might be replaced by José M. Martín-García's Mathematica package [XTensor](#) in the future (free, going to be released soon)

Tools for the transition from abstract index notation to explicit eqs. implemented in the code: 2 versions

- based on MathTensor
- based on Ian Hinder's [TensorTools](#)

Scripts for generating code evolution system first order in time + associated equations, given in MathTensor syntax \Rightarrow generate Cactus thorns.

Examples Klein Gordon, Maxwell, ADM, BSSN, MCW, CFE

Kranc's Tensor Tools

define additional structure on top of MathTensor in form of rules and functions.

3+1 decomposition: automatized to large extent

- specify spatial variables
- call functions that define projection rules etc.
- define rules specific to system
- apply rules
- **Examples:** Maxwell, ADM (\rightarrow BSSN), MCW, CFE

Propagation of the Constraints

Constraints written as expressions like

$$Ch_{abc} - \nabla_c h_{ab}$$

where Ch_{abc} is defined as a tensor symmetric in its first two indices.

constraint satisfied $\Leftrightarrow Ch_{abc} = 0$.

- define rules, e.g. $\partial_t K_{ab} \rightarrow \mathcal{L}_\beta K_{ab} + \alpha \nabla_p \gamma^p_{ab} + \dots$ (automatized)
- take time derivatives of constraints, use time evolution equations
- replace spatial derivatives by constraints
- if result is not homogeneous in constraints: figure out why/whether inhomogeneous terms have to vanish (might be most tedious part)
- **Examples:** Maxwell, MCW, CFE

Characteristics

H. Friedrich, *Hyperbolic reductions for Einstein's eqs*, Class. Quantum Grav. **13** (1996) 1451 – 1469

J. Stewart, *The Cauchy problem and the IBV problem in NR*, Class. Quantum Grav. **15** (1998)

- for first order evolution system:

$$\partial_t \mathbf{u} = \mathbf{A}^k \partial_k \mathbf{u} + l.o.,$$

where $\mathbf{u} = (h_{ab}, K_{ab}, \gamma^c_{ab}, \dots, \dots)^T$

- extract principal part (**automatized**), $\partial_a \rightarrow \xi_a$, $\partial_t \rightarrow \xi_0$ (**automatized**),
- characteristic speeds and fields: compute (ξ_0, \mathbf{u}) such that

$$\mathbf{A}^k \xi_k \mathbf{u} = -\xi_0 \mathbf{u},$$

- stay on level of abstract indices throughout

- Solve system **manually** by case distinction:
 - $\xi_0 = S^k \xi_k$ ($n^\mu \xi_\mu = 0$) \Rightarrow compute eigenvectors
 - $\xi_0 \neq S^k \xi_k \Rightarrow$ reduced system, new candidate for eigenvalue pops up
 - repeat
- **Kranc** tools support algebraic manipulations.
- **Examples:** MCW, CFE and evolution system of constraints (principal parts very simple)

Under Construction: Frames

H. Friedrich and G. Nagy, *The initial boundary value problem for Einstein's vacuum field equations*, Commun. Math. Phys. **201**, 619 (1999)

- implement system numerically
- variables: $e_i^\mu, \Gamma_{i^j k}, C_{ijkl}$
- extend mathematica scripts to handle *frames*: [Dana Alic](#)

Generating Code

Sascha, Ian

Evolution system, first order in time + other equations

⇒ generate complete Cactus thorns (evolution based on CactusMoL)

Preparations:

Transform equations involving 3-d tensors (in MathTensor syntax) to equations for variables used in the code

- convert covariant/Lie derivatives to ordinary derivatives
- expand sums over pairs of indices, split equations into individual components
- name variables and partial derivatives, e.g. $h[-1, -1] \rightarrow h11$,
 $OD[h[-2, -2], -1] \rightarrow D1[h22]$

there are two possibilities to do this:

- rules on top of MathTensor
- Ian's package [TensorTools.m](#) replaces MathTensor for this purpose.

Automatic Generation of Cactus Thorns – Tools

set of Mathematica packages, designed for generating Cactus thorns, but only small part of functionality knows about Cactus \Rightarrow could be modified to suit other software infrastructure.

- written in the style of “functional programming”
- clear structure, readable, documented
- flexible
- produce readable and documented code

Packages are e.g.

KrancThorns.m loads packages that are needed, contains user level functions for generating specific thorns (e.g. Base, MoL)

Thorn.m functions to create various parts of a Cactus thorn

CodeGen.m functions for constructing code

Parfiles.m functions for generating parameter files for Mexico tests

Further Features: interfaces to

- standard Cactus boundary conditions
- excision

Planned: interfaces to

- Whisky
- Erik's elliptic solver

Ongoing

- write paper, including documentation of the main features and examples

Automatic Generation of Cactus Thorns – User level

generate complete Cactus thorns:

- interface.ccl, params.ccl, schedule.ccl, source files

Type of Kranc thorns :

Base: define grid functions, initialize symmetries

MoL: register variables for MoL, set MoL rhs variables, boundary conditions, excision

Translate: translate between Cactus variables and evolution variables (e.g. set initial data)

Evaluate: define new grid functions, set them (e.g. evaluate constraints)

Set: set “primitives” (grid functions, that are not evolved), e.g. gauge

generate ThornList

Finite Differencing :

Finite difference scheme is implemented in separate thorn **GenericFD:**

at the moment macros for

- second order centered differences
- fourth order centered differences

User specifies:

- equations in lists (evolution equations, constraints,)
- evolution variables (grid functions, evolved),
“primitives” (grid functions, not evolved),
“shorthands” (abbreviations, not grid functions),
parameters
- criteria for simplification (“CollectList”)
- language (C, Fortran)

Example notebooks and scripts :

Klein Gordon, Maxwell, ADM, BSSN, CFE, MCW

code generated this way \Rightarrow Mexico tests

Example – The ADM Equations

Evolution equations

$$\partial_t h_{ab} = \mathcal{L}_S h_{ab} + 2 A K_{ab}$$

$$\partial_t K_{ab} = \mathcal{L}_S K_{ab} + D_a D_b A + 2 A h^{pq} K_{pb} K_{qa} - A K K_{ab} - A R_{ab}$$

Constraints

$$R - h^{pq} h^{rs} K_{pr} K_{qs} + K^2 = 0$$

$$h^{pq} D_p K_{aq} - D_a K = 0,$$

where

$$\begin{aligned} R_{ab} = & h^{pq} (-\partial_a \partial_b h_{pq} + \partial_b \partial_q h_{pa} + \partial_a \partial_p h_{qb} - \partial_p \partial_q h_{ab})/2 \\ & + h_{pr} h^{qs} (\gamma^p_{qa} \gamma^r_{sb} - \gamma^p_{ab} \gamma^r_{qs}) \end{aligned}$$

Base Thorn :

- specify groups of **evolution variables**

```
ADMvars = {{ "h", {h11,h21,...} }, {"K", {K11,K21,...} } }
```

- specify groups of **primitives** e.g.

```
ADMprim = { {"A", {A}}, {"S", {S1,S2,S3}}, {"trK", {trK}} }
```

- **create Base Thorn:**

```
createBaseThorn[ADMvars, ADMprim,
  SystemDescription -> "ADM",
  Formulation -> "KrancADM"]
```

Main Evolution Thorn :

- specify calculation

```

MainMoLCalculation = {
  Shorthands      -> {hInv11,..., gamma111,...,R11,...},
  CollectList     -> {hInv11,...},
  GridFunctions   -> {h11,..., K11,...,h11rhs,.....},
  Equations       -> {{hInv11      -> ..., .....,
                       gamma111   -> ..., .....,
                       R11         -> ..., .....,
                       dot[h11]    -> ..., .....,
                       dot[K11]    -> ...,.....}}}}

```

- create MoL Thorn:

```

createMoLThorn[MainMoLCalculation,ADMvars,
  PrimitiveGroups -> ADMprim,
  Thorn            -> "KrancADMMainEvol",
  Implementation  -> "KrancADMMainEvol"]

```

Translator Thorn

- cactus variables \rightarrow evolution variables at POSTINITIAL
- evolution variables \rightarrow cactus variables in MoL_PostStep

Evaluator Thorn evaluate constraints

- specify **Calculation**: (as before) Shorthands, Gridfunctions, CollectList, Eqs
- create **Evaluator Thorn** (as before)

```
createEvaluateThorn[
    {{ "CH" , HamCalculation } , { "CM" , MomCalculation } } ,
    EvaluationGroups ,
    Thorn -> . . . . ,
    Implementation -> . . . . . ,
    Formulation -> . . . . . ]
```

Setter Thorns : e.g. Lapse, Shift, trK

- set **shift** \rightarrow 0: specify Calculation, create SetThorn:

```
createSetThorn[ZeroShiftCalculation,  
  Groups    -> ...,  
  Thorn      -> ...,  
  SetTime   -> "initial_and_poststep",  
  Formulation -> ...]
```

- set [harmonic lapse](#) (as above)
- set [trK](#) (as above)

create Thornlist information on created thorns and thorns they depend on, returned by functions `create...Thorn`