Arbogast: "Du calcul des dérivations" to higher order AD

Isabelle Charpentier[a] & Jens Gustedt[a,b]
[a]Icube, UMR 7357, CNRS&University de Strasbourg
[b]INRIA, France

arbogast: https://hal.inria.fr/hal-01307750
Modular C: https://hal.inria.fr/hal-01169491

Table of Contents

Louis François Antoine Arbogast
Mathématicien révolutionnaire (1759–1803)

1786 Prix de l'Académie de Mantoue
"Mémoire sur les principes généraux de la Mécanique" en 1786

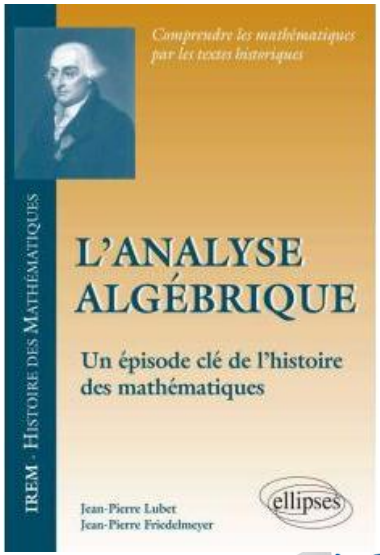
1789 Prix de l'Académie de Saint Pétersbourg
"Mémoire sur la nature des fonctions arbitraires qui entrent dans les intégrales des équations différentielles partielles"

1789 "Mémoire sur de nouveaux principes de calcul différentiel indépendants de la théorie des infiniment petits"
(présenté à l'Académie royale des sciences,Paris)

En politique de 1789 à 1795,
Collection de manuscrits et textes scientifiques
(dont copies des textes de Fermat)

1795 Direction de l'Ecole centrale de Strasbourg

1800 "Du calcul des dérivations"



J.-P. Friedelmeyer (2016). Arbogast, de l'Institut national de France.
https://hal.archives-ouvertes.fr/hal-01326311

Louis François Antoine Arbogast dans la révolution 1789→1795

1790 Conseil général de Strasbourg

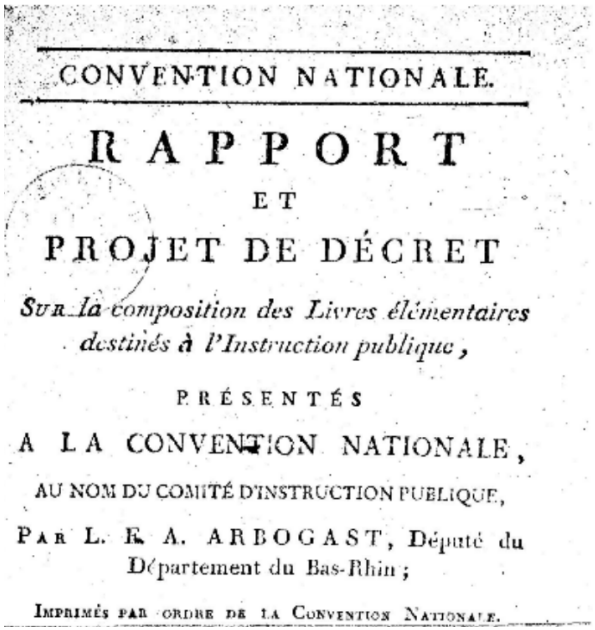
1791 Assemblée Législative, député du département appelé Bas-Rhin

1792 Convention nationale, montagnard modéré

Comité d'instruction publique de la Convention nationale

1792 Rapport et projets de décret sur la composition des livres élémentaires destinés à l'instruction publique

1793 Rapport sur l'uniformité et le système général des poids et mesures



CONVENTION NATIONALE.

RAPPORT

ET

PROJET DE DÉCRET

Sur la composition des Livres élémentaires destinés à l'Instruction publique,

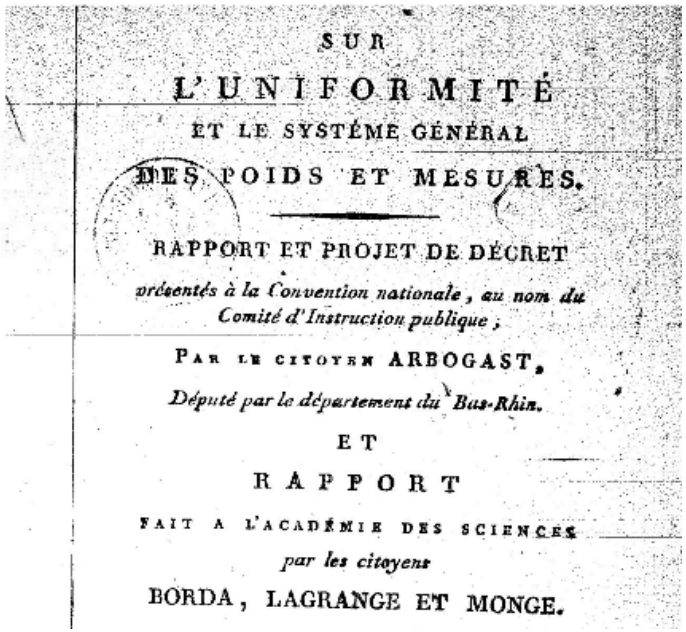
PRÉSENTÉS

A LA CONVENTION NATIONALE,

AU NOM DU COMITÉ D'INSTRUCTION PUBLIQUE,

PAR L. F. A. ARBOGAST, Député du Département du Bas-Rhin;

IMPRIMÉS PAR ORDRE DE LA CONVENTION NATIONALE.



SUR

L'UNIFORMITÉ

ET LE SYSTÈME GÉNÉRAL

DES POIDS ET MESURES.

RAPPORT ET PROJET DE DÉCRET

présentés à la Convention nationale, au nom du Comité d'Instruction publique;

PAR LE CITOYEN ARBOGAST,

Député par le département du Bas-Rhin.

ET

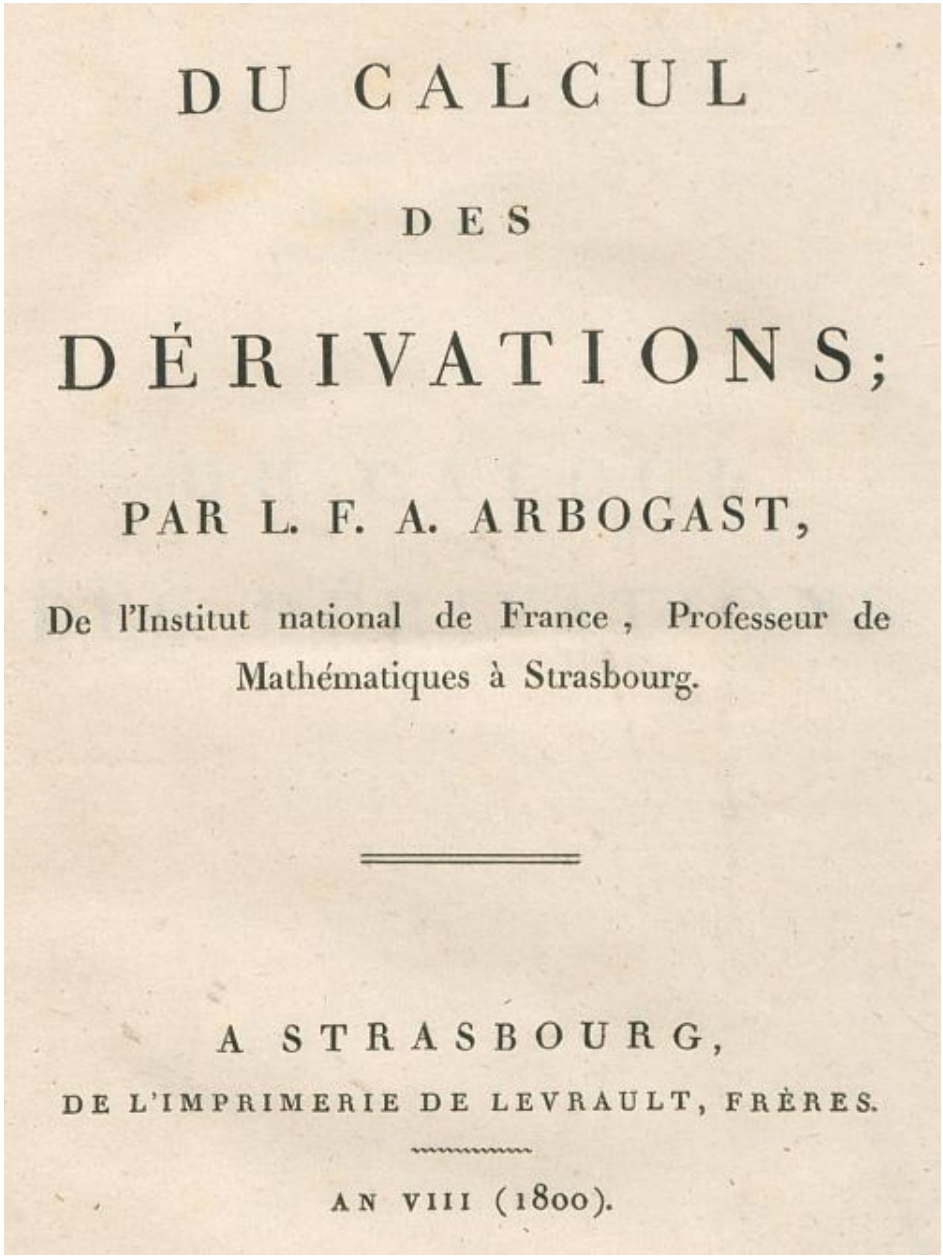
RAPPORT

FAIT A L'ACADÉMIE DES SCIENCES

par les citoyens

BORDA, LAGRANGE ET MONGE.

1795 Retour à Strasbourg



DU CALCUL

DES

DÉRIVATIONS;

PAR L. F. A. ARBOGAST,

De l'Institut national de France, Professeur de Mathématiques à Strasbourg.

A STRASBOURG,

DE L'IMPRIMERIE DE LEVRAULT, FRÈRES.

AN VIII (1800).

http://docnum.u-strasbg.fr/cdm/ref/collection/coll7/id/40373

Préambule

Cette méthode de calcul est fondée sur une manière générale de considérer les quantités comme dérivant les unes des autres ; c'est ce qui me l'a fait nommer *Calcul des dérivations*.

les dérivées
que je considère sont moins des dérivées de quantités que des dérivées
d'opérations

Pour former l'algorithme des dérivations, il a fallu introduire des signes nouveaux ; j'ai donné une attention particulière à cet objet, persuadé que

In a nutshell

Arbogast's work concerns the HO derivation of

$$\varphi \circ \psi(x)$$

assuming $\psi(x)$ is a Taylor polynomial

$$\psi(x) = \psi_0(x) = \sum_m \psi_m x^m \qquad \text{where} \qquad \psi_m = \frac{1}{m(m-1)..1}\psi^{(m)}(x)$$

Arbogast

- makes use the notion of factorial, and partitions
- introduces "différentielles divisées" i.e. Taylor coefficients ψ_m
- predates Faà di Bruno's formula (57 years)
- exemplifies his work on elemental functions

Generalized (higher order) chain rules

1800

Arbogast's rule for the higher order (n°41, p 34)

$$\underset{C}{\mathrm{D}}^{m}.\varphi\psi_0 = \underset{C}{\mathrm{D}}^{m}\varphi\psi_0.\psi_1^m + \underset{C}{\mathrm{D}}^{m-1}\varphi\psi_0.\mathrm{D}.\psi_1^{m-1} + \underset{C}{\mathrm{D}}^{m-2}\varphi\psi_0.\underset{C}{\mathrm{D}}^{2}.\psi_1^{m-2} + \ldots + \underset{C}{\mathrm{D}}\varphi\psi_0.\underset{C}{\mathrm{D}}^{m-1}.\psi_1$$

is recursive and runs from $\varphi\psi_0$ called "origine des dérivations", details in n°52, p 43

- but little known, except in Great Britain: Woodhouse, Babbage, Cayley, Sylvester, Horner, ...

1857

Faà di Bruno's formula

$$D_x^m\varphi = \sum \frac{\Pi(m)}{\Pi(i)\,\Pi(j)\cdots\Pi(k)} D_y^p\varphi.\left(\frac{\psi'}{1}\right)\left(\frac{\psi''}{1.2}\right)\left(\frac{\psi'''}{1.2.3}\right)\cdots\left(\frac{\psi^{(l)}}{\Pi(l)}\right)$$

- known formula, and known controversy [Johnson, Craik]

Higher order derivation of the exponential function (pages 10&11)

EXEMPLE II.

14. On demande à développer la fonction

$$e^{\alpha} + \mathcal{C}x + \gamma x^2 + \delta x^3 + \varepsilon x^4 + \zeta x^5 + \text{etc.},$$

$\log e$ étant $= 1$, en une série de cette forme :

$$A + Bx + Cx^2 + Dx^3 + Ex^4 + Fx^5 + \text{etc.}$$

15. Si l'on veut calculer les coëfficiens A, B, C, D etc. en termes récurrens, on les trouvera avec la plus grande facilité.

On a $A = e^{\alpha}$; d'où par dérivation $B = e^{\alpha}\mathcal{C}$, et, en mettant A au lieu de e^{α}, $B = A\mathcal{C}$. Il suffira de continuer les dérivations, et l'on aura

$$A = e^{\alpha};$$
$$B = A\mathcal{C};$$
$$2C = A\,2\gamma + B\mathcal{C},$$
$$C = A\gamma + \tfrac{1}{2}B\mathcal{C};$$
$$3D = A3\delta + B\gamma + \tfrac{1}{2}B2\gamma + \tfrac{1}{2}.2C\mathcal{C},$$
$$D = A\delta + \tfrac{2}{3}B\gamma + \tfrac{1}{3}C\mathcal{C};$$
$$4E = A4\varepsilon + B\delta + \tfrac{2}{3}B3\delta + \tfrac{2}{3}.2C\gamma + \tfrac{1}{3}C2\gamma + \tfrac{1}{3}.3D\mathcal{C},$$
$$E = A\varepsilon + \tfrac{3}{4}B\delta + \tfrac{2}{4}C\gamma + \tfrac{1}{4}D\mathcal{C};$$

etc., où la loi est manifeste.

Higher order derivation formulas

Arithmetic operations		Elemental fuctions		
$v =$	Recurrence $(v_k =)$	$v =$	Recurrence	Reference in Arbogast's book
$u + w$	$u_k + w_k$	$\ln(u)$	$\tilde{v}_k = \frac{1}{u_0}\left(\tilde{u}_k - \sum_{i=1}^{k-1} \tilde{v}_j * u_{k-j}\right)$	$n°12$, p 6
$u * w$	$\sum_{i=0}^{k} u_j * w_{k-j}$	$\exp(u)$	$\tilde{v}_k = \sum_{i=1}^{k} \tilde{u}_j * v_{k-j}$	$n°14$, p 10
u/w	$\frac{1}{w_0}\left(u_k - \sum_{i=0}^{k-1} v_j * w_{k-j}\right)$	$\sin(u)$	$\tilde{s}_k = \sum_{i=1}^{k} \tilde{u}_j * c_{k-j}$	$n°16$, p 11
$\sqrt{u}$	$\frac{1}{2v_0}\left(u_k - \sum_{i=1}^{k-1} v_j * v_{k-j}\right)$	$\cos(u)$	$\tilde{c}_k = \sum_{i=1}^{k} -\tilde{u}_j * s_{k-j}$	$n°32$, p 27

Can be found in [Griewank&Walther, 2008]

It's an easy job to implement these HOAD recurrence formulas by means of operator overloading

Table of Contents

Experience as AD user and HOAD developper

Modes

Tangent/adjoint → source to source transformation of Fortran codes
HOAD → develop of my own tools for specific applications

Best practices

Mode	Application	Software	Co-authors	Year
Adjoint	Meteorological model MESO-NH	"Odyssée"	Ghémirès	1998
HOAD	Higher order continuation	Diamant	–	2008
HOAD	Mixed derivatives in quantum chemistry	Rapsodia	Utke	2009
HOAD	Nonlinear mechanics	Diamanlab	Cochelin/Lampoh	2013
HOAD	Special functions (Bessel, ...) Optics	Arbogast	Gustedt	2018

+ a certain number of applications managed using Tapenade [Hascoët&Pascual, 2013]

Some general higher order frameworks for nonlinear problems

Some frameworks	Authors	Some details
ATOMFT	Corliss & Chang	ODEs and DAEs
COSY INFINITY	Berz & Makino	Taylor models (HOAD+interval)
DAETS	Pryce & Nedialkov	DAEs
The Taylor Center	Gofen	ODEs
Diamant	Charpentier	Higher order continuation

Nonlinear problems

Problems	General formula	Some applications
Boundary value	$\mathcal{R}(u,\lambda)=0$	NL behaviour laws and/or NL forced vibrations in mechanics Robotics, Shape optimization
Eigenvalue	$\mathcal{P}(\lambda)u=0$	NL eigenproblems
	$\mathcal{Q}(u,\lambda)u=0$	NL free vibration
ODE / DAE	$\mathcal{R}(u,\frac{\partial u}{\partial t},\lambda)=0$	Pendulum, Plasticity, Friction
PDE	$\mathcal{R}(u,\frac{\partial u}{\partial t})=0$	Instationary PDE problems (fluids or structures)

[Charpentier, Optim. methods Soft., 2012] for references

Table of Contents

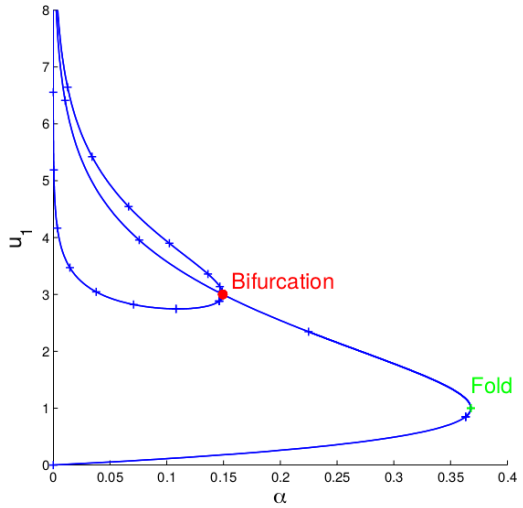
Tutorial example

Nonlinear parametric equations (from [MatCont])

$$\mathcal{R}(u_1, u2, \lambda) = \begin{cases} 0 = -2u_1 + u_2 + \lambda \exp(u_1), \\ 0 = u_1 - 2u_2 + \lambda \exp(u_2). \end{cases} \quad (1)$$

Unknown	$u(\lambda) = (u_1(\lambda), u_2(\lambda))$
Parameter	λ (varies)
Trivial solution	$(u_1(\lambda), u_2(\lambda), \lambda) = (0, 0, 0)$

Bifurcation diagram



Goal

Compute solution branches $(u_1(\lambda), u_2(\lambda), \lambda)$ for $\lambda \geq 0$

Higher-order continuation with Diamant [Charpentier 2012]

General nonlinear residual problem

$$\mathcal{R}(\mathbf{U}(a)) = 0$$

where $\mathbf{U} = (u, \lambda)$, u is the unknown state vector and λ a scalar parameter to be varied

+

pseudo-arclength equation to close this under-determined system [Keller, 1987]

$$a = \left\langle \mathbf{U}(a) - \mathbf{U}(0), \frac{\partial \mathbf{U}}{\partial a}(0) \right\rangle$$

Approximations through truncated Taylor series (under analitycity assumptions)

$$\begin{cases} \mathbf{U}(a) = \sum_{k=0}^{K} a^k \mathbf{U}_k \quad \text{where} \quad \mathbf{U}_k = \frac{1}{k!} \frac{\partial^k \mathbf{U}}{\partial a^k}(0) \\ \mathcal{R} \circ U(a) = \sum_{k=0}^{K} a^k \mathcal{R}_k \end{cases}$$

Diamant's nonlinear solver

HOAD nonlinear solver (under analitycity assumptions)

- Inject $U(a) = \sum_{k=0}^{K} a^k \mathbf{U}_k$ into $\mathcal{R}(\mathbf{U}(a)) = 0$
- Use analitycity and Arbogast's rule
- Deduce

$$\{\mathcal{R}_k\} = \left\{\mathcal{R}_{k|\mathbf{U}_k=\mathbf{0}}\right\} + \{\mathcal{R}_1\}\,\mathbf{U}_k = 0, \qquad \text{for all } k = 1,..,K$$

- then

$$\begin{cases} \{\mathcal{R}_1\}\,\mathbf{U}_1 = -\left\{\mathcal{R}_{1|\mathbf{U}_1=\mathbf{0}}\right\}, & \langle \mathbf{U}_1, \mathbf{U}_1 \rangle = 1 \\ \{\mathcal{R}_1\}\,\mathbf{U}_k = -\left\{\mathcal{R}_{k|\mathbf{U}_k=\mathbf{0}}\right\}, & \langle \mathbf{U}_k, \mathbf{U}_1 \rangle = 0, \quad \text{for } k = 2,..,K \end{cases}$$

→ **Solve the general nonlinear problem as an iterative sequence of linear systems**

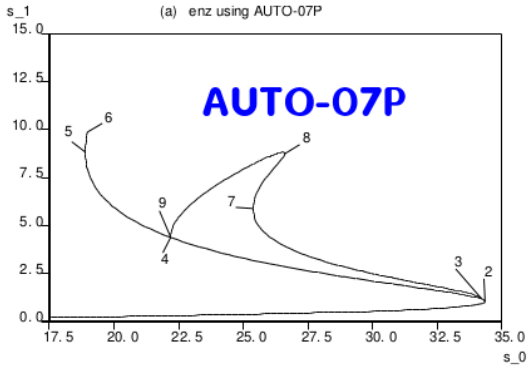
→ Use of AD to evaluate the Jacobian $\{\mathcal{R}_1\}$ and higher order terms $\left\{\mathcal{R}_{k|\mathbf{U}_k=\mathbf{0}}\right\}$ (Similar arguments in the other frameworks)

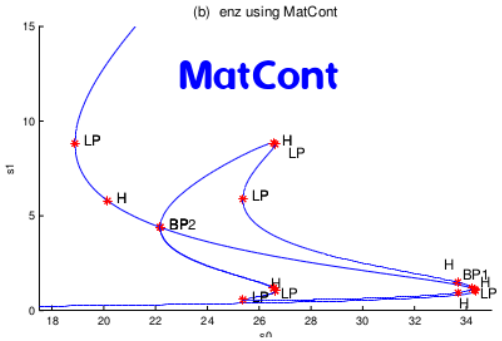
Diamant's framework agrees with homotopy, sensitivity computations, workspace boundary calculation, bifurcation analysis, combination of former methods, ...

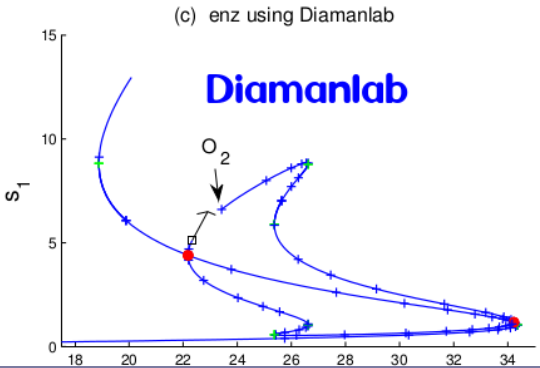
Diamanlab: a Matlab implementation of Diamant

[IC, Cochelin&Lampoh, 2013]

Bifurcation diagram of the enzyme model







Two-compartment enzyme model [Kernevez,1980]

$$\begin{cases} s_1' = (s_0 - s_1) + (s_2 - s_1) - \dfrac{100 s_1}{1 + s_1 + s_1^2}, \\ s_2' = (s_0 - s_2) + (s_1 - s_2) - \dfrac{100 s_2}{1 + s_2 + s_2^2}, \end{cases}$$

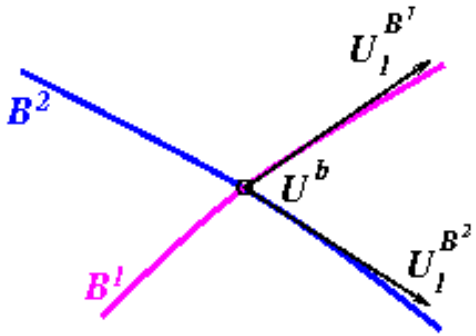
Diamanlab's advantages: Large steps, accurate plots

B1	Type	Label	AUTO	MatCont	Diamanlab
	LP	2	141	152	8
	H	–	–	155	–
	BP1	3	145	157	10
	BP2	4	214	211	18
	H	–	–	224	–
	LP	5	244	243	19

Drawback: Matlab's interpreted language

Bifurcation analysis

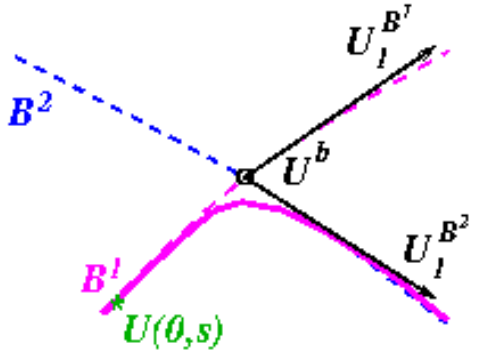
Perform from rank deficiency of the Jacobian at bifurcation points



Bif. detection

(1st order equation)
Rank deficiency of the Jacobian

- Sign of $\det(\mathcal{R}_1)$
- Geometric series
  [van Dyke 1974]
  [Cochelin&Médale 2013]



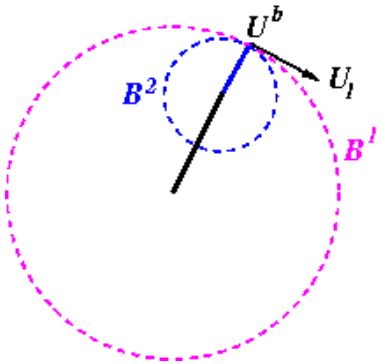
Branch switching

(2nd order equation)

- Tangents from ABE
  [Doedel et al. 1991]
- Series
  [Cochelin&Médale 2013, AD2016]

Other options

- Pertubation
- Deflation **[Farrell et al. 2016]**



Curvature

(3rd order equation)

- Direct solution
- (Many critical cases in robotics)

Other options

- Approximate curvature
- Perturbation

Elastica [Doedel et al. 1991; Cochelin&Médale 2013; Farrell et al. 2016]

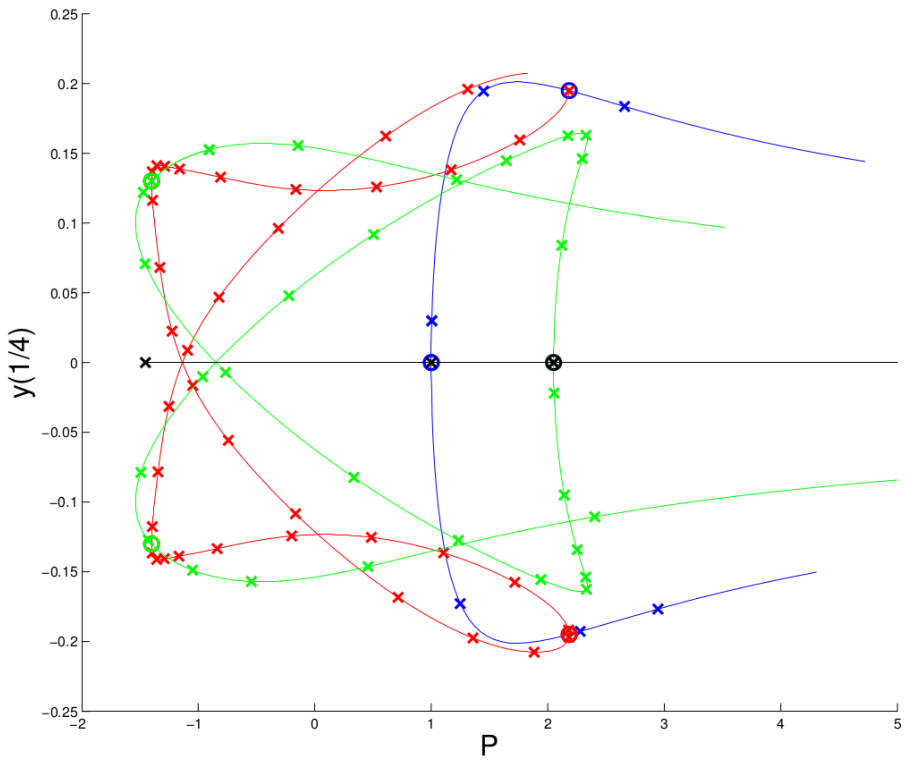
Equations for beam equilibriums

$$\begin{cases} x'(s)=\cos(\theta(s)), \\ y'(s)=\sin(\theta(s)), \\ \theta'(s)=m(s), \\ m'(s)=-4\pi^2(P\sin(\theta(s))+T\cos(\theta(s)), \end{cases} \quad (2)$$

Clamped-Clamped case

Discretization

- Regular mesh of N_e elements
- cubic interpolation for the unknowns
- Orthogonal colloc. method at Gauss points
- About $3 \times N_e \times 4$ equations (55 here)



Case study in AUTO and Diamanlab
Not really possible in Matcont

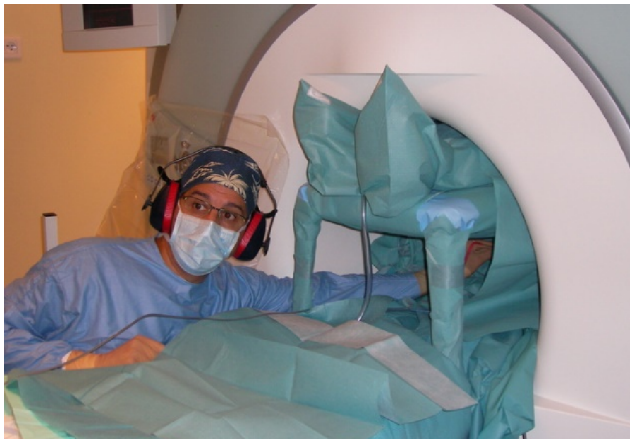
Download http://icube-web.unistra.fr/cs/index.php/Software_download#Diamanlab

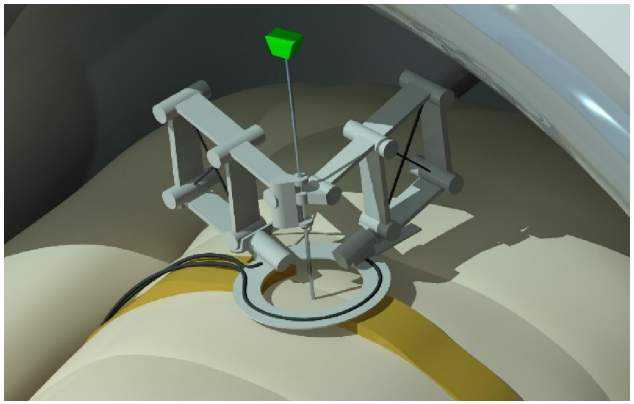
Table of Contents

Robotics

Medical and surgical context

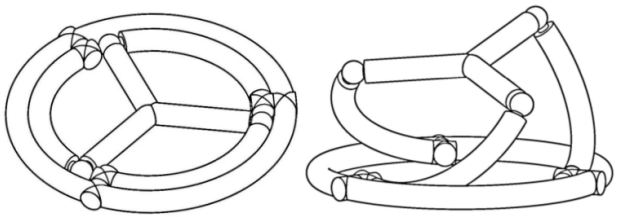
- Magnetic Resonance Imaging
- Robots for safer procedures
- Need for compact architectures





Objectives

- Optimized mechanisms
- MRI compliant materials
- Numerical tools



Higher order continuation for robot analysis & workspace determination
[Hentz, Renaud, Rubbert, IC]

Augmented system for workspace boundary for stability domain, [Seydel 1979] or robot workspace [Litvin 1980])

Workspace boundary

$$\partial\mathcal{W} = \{\mathbf{u} \in \mathcal{W} \mid \exists\ (\xi, \mathbf{v}) \quad \text{s.t.} \quad \left(\mathcal{R}^{\mathcal{W}}_{\mathbf{v}}\right)^{\top}(\mathbf{u}, \mathbf{v}).\xi = 0 \text{ and } \xi^{\top}.\xi = 1\}$$

Implementations

- "Math version": MATCONT, for ODE and stab. domains [Govaerts&Kuznetzov 2007] uses AD
- "Robotic version"
  - Handcoded Jacobian in [Haug et al. 1996] + First order continuation
  - Source transformation [Hentz et al. 2016] + Higher order continuation

$$\mathcal{R}^{\mathcal{W}}(\mathbf{u}, \mathbf{v}) \xrightarrow[\text{system}]{\text{Augment}} \begin{pmatrix} \mathcal{R}^{\mathcal{W}}(\mathbf{u}, \mathbf{v}) \\ \left(\mathcal{R}^{\mathcal{W}}\right)^{\top}(\mathbf{u}, \mathbf{v}).\xi \\ \xi^{\top}.\xi - 1 \end{pmatrix} \xrightarrow[\text{w.r.t. } \mathbf{v}]{\text{Diff.}} \begin{pmatrix} \mathcal{R}^{\mathcal{W}}(\mathbf{u}, \mathbf{v}) \\ \mathcal{R}^{\mathcal{W}}_{\mathbf{v}}(\mathbf{u}, \mathbf{v}) \\ \left(\mathcal{R}^{\mathcal{W}}\right)^{\top}(\mathbf{u}, \mathbf{v}).\xi \\ \left(\mathcal{R}^{\mathcal{W}}_{\mathbf{v}}\right)^{\top}(\mathbf{u}, \mathbf{v}).\xi \\ \xi^{\top}.\xi - 1 \end{pmatrix} \xrightarrow[\textit{information}]{\textit{Pick up}} \mathcal{R}^{\partial\mathcal{W}}(\mathbf{u}, \mathbf{v},$$

Solution for $\mathcal{R}^{\partial\mathcal{W}}(\mathbf{u}, \mathbf{v}, \xi)$ through higher order continuation

Workspace boundaries of the RR planar arm

Forward kinematics for the RR

$$\begin{cases} x = l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) \\ y = l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2) \end{cases}$$

Joint constraints

$\theta_i = \sin(\frac{\pi}{3} v_i)$, for $v_i \in \mathbb{R}$

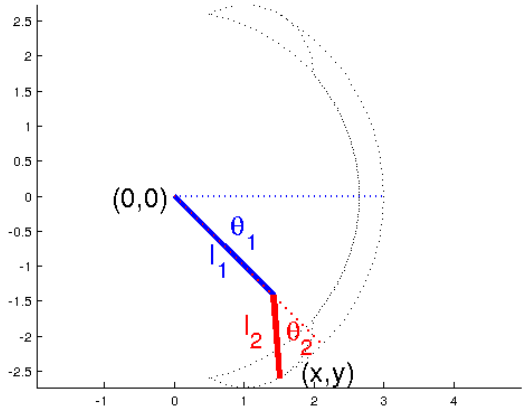
RR Implementation

$$\begin{array}{cccccc} \mathcal{R}^{\mathcal{W}}: & \mathbb{R}^2 & \rightarrow & [-\frac{\pi}{3}, \frac{\pi}{3}]^2 & \rightarrow & \mathcal{W} \\ & \mathbf{v} & \mapsto & (\theta_1, \theta_2) & \mapsto & \mathbf{u} = (x, y) \end{array}$$

Workspace

$$\mathcal{W} = \left\{ \mathbf{u} \mid \exists\ \mathbf{v} \quad \text{s.t.} \quad \mathcal{R}^{\mathcal{W}}(\mathbf{u}, \mathbf{v}) = 0 \right\}$$

$\mathcal{R}^{\partial\mathcal{W}(\mathbf{v},\mathbf{u},\xi)}$ **implementation**
Then continuation with no favoured parameter

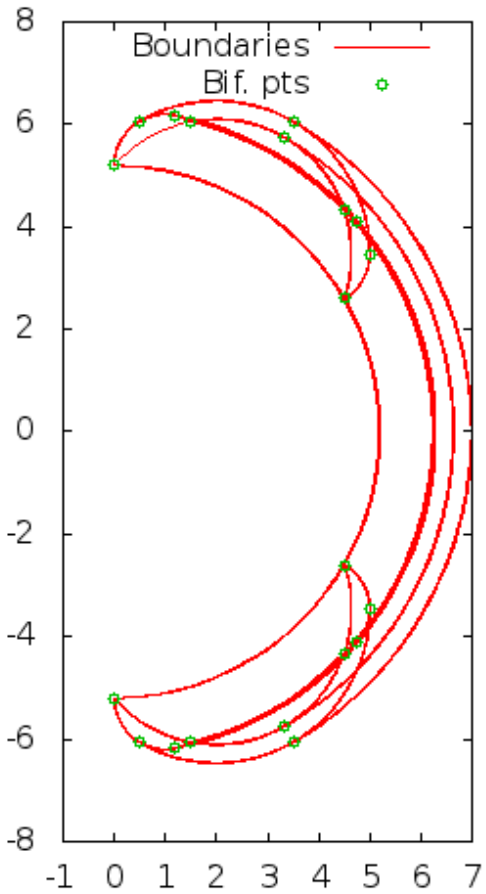


Critical problems

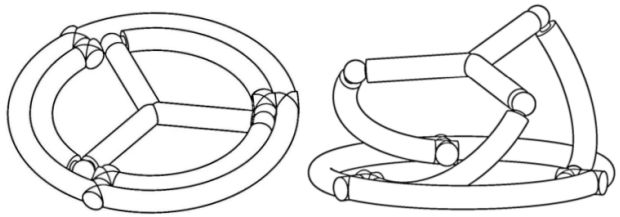
- 2 angular sets for **u** (elbow up, elbow down)
- Infinite number of **v** for **u**
- Bifurcation with same tangent but different curvatures

Some other workspaces

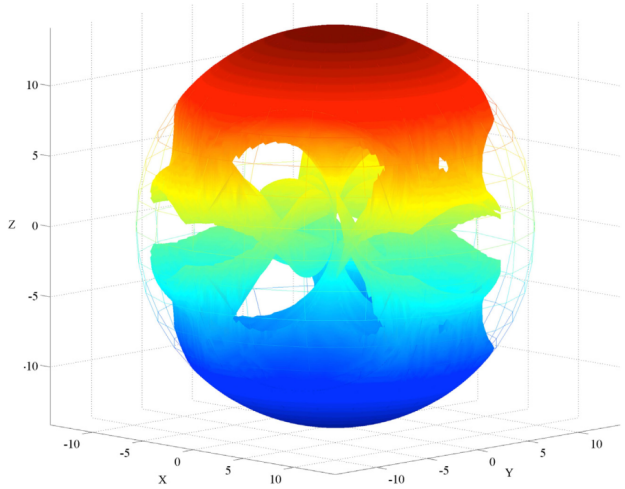
RRR's workspace (implemented in F90)



3US mechanism



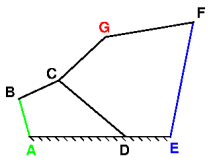
3US workspace



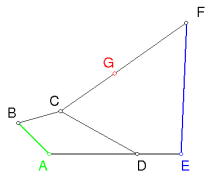
Classification of singularities

Classification of singularities from rank deficiency of Jacobians (21 classes)

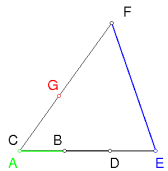
[Zlatanov et al. 1998], operated with AD in [Hentz et al. 2016]



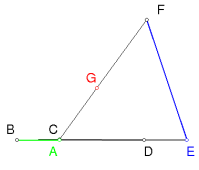
6 bars



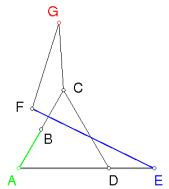
Forward singularity (1)



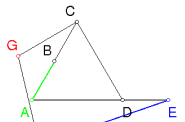
Singularité (2)



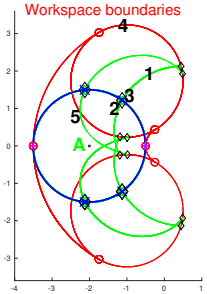
Singularité (3)



Inverse singularity (4)



Inverse singularity (5)



Bifurcation diagram

Options for planar workspaces

- Higher order series for bifurcation detection
- Jacobian calulation for classification
- Perturbation method for branch switching

Options for 3D workspaces

- Planar mechanism workspace or 2D projections
- Implementing Henderson's method for surface evaluation

Table of Contents

From Faddeeva's function to arbogast

Faddeeva function (originally Kramp function!)

- scaled complex complementary

$$w(z) = e^{-z^2}\left(1 + \frac{2i}{\sqrt{\pi}}\int_0^z e^{t^2}\,dt\right) = e^{-z^2}\operatorname{erfc}(-iz). \tag{3}$$

- satisfies the recurrence formula

$$\begin{gathered} w^{(k+2)}(z) + 2zw^{(k+1)}(z) + 2(k+1)w^{(k)}(z) = 0 \\ (\text{for } k = 0:\ \alpha(z) = 1,\ \beta(z) = 2z \text{ and } \gamma(z) = 2) \end{gathered} \tag{4}$$

- reccurence seed

$$w^{(0)}(z) = w(z), \qquad w^{(1)}(z) = -2zw(z) + (2i/\sqrt{\pi}) \tag{5}$$

More generally, **the general second order ordinary differential equation**

$$\alpha(z)\varphi^{(2)}(z) + \beta(z)\varphi^{(1)}(z) + \gamma(z)\varphi^{(0)}(z) = 0$$

can be used for optimized HOAD of special functions from known seeds $\varphi^{(1)}(z)$ and $\gamma(z)\varphi^{(0)}$ [Charpentier, 2015]

Functions of mathematical physics [Abramowitz&Stegun, DMLF]

Some examples among many others

		Seed			2nd order ODE			Validation		
Name	φ_n	a_n	b_n	g_n	α_n	β_n	γ_n	A_n	B_n	G_n
Bessel	J_n	1	$-\frac{n}{z}$	1	z^2	z	$(z^2 - n^2)$	1	$\frac{2n}{z}$	1
Cheb.	T_n	$1 - z^2$	$-nz$	n	$1 - z^2$	$-z$	n^2	1	$(2 - \delta_{n,0})z$	1
Gauss	${}_2F_1$	$z(1-z)$	$n - \xi + mz$	$\xi - n$	$z - z^2$	...	$-n\mu$	...	...	$\xi - n$???

There is a need for automatic generation of HOAD libraries

- to account for operand types and recurrence formulas
- to implement HOAD of special functions

Examples

- Rapsodia [Charpentier&Utke, 2009] is a python-based generator for C++ and F90 HOAD libraries
  that unrolls recurrence loops
- `arbogast` [IC&Gustedt, 2018] an HOAD library based on Modular C

Modular C summary

C: Lack of modularity, reusability and encapsulation.

- programming large projects in C is a pain
- current main strategy: naming conventions
- the naming conventions of the C standard
  - intrusive, e.g `strip` and `top` are reserved
  - inconsistent, growing, incomplete, ...

Modular C: `http://cmod.gforge.inria.fr/`

- an existing reference implementation
- open source
- mostly operational

Modular C summary

Modular C is an extension of C

- an hierarchical naming convention for modules
- an abbreviation feature to ease the pain
- snippets for code reuse
- foreach for compile time code unrolling

Other features flow from these major features

- an acyclic import graph
- painless module initialization and shut down
- a comprehensive structure of the C library
- interchangeability of modules
- reusability of software components
- better code factorization

arbogast, a Modular C package

arbogast provides Taylor polynomial types

- six Taylor types $P(t) = x_0 + x_1 t + x_2 t^2 + \cdots + x_N t^N$
- encoded by the coefficients $x = (x_0, x_1, \ldots, x_N)$

arbogast provides functions for Taylor polynomials

function	result
$x = (x_0, x_1, \ldots)$	$\approx Q(t) = r_0 + r_1 t + \cdots$
arbogast::eval(x, t)	evaluation in point t, $x_0 + x_1 t + x_2 t^2 + \cdots$
arbogast::add(x, y)	polynomial sum, $(x_0 + y_0, x_1 + y_1, \ldots, x_N + y_N)$
arbogast::sin(x)	$\sin(x_0 + x_1 t + x_2 t^2 + \cdots)$

arbogast provides "context" for Taylor polynomials

« $((\kappa\text{-}1)*\rho + x/\rho\kappa)/\kappa$ » $\Longrightarrow$ div(prod(minus(κ,1), ρ) + div(x,$\rho\kappa$), κ) }

arbogast provides differential operators

- special functions as solutions of the general 2nd order ODE

Example, C

Heron's method for the approximation of $\sqrt[kappa]{x}$

```
double phi(double x, unsigned kappa) {
  if (kappa == 1) return x;                          // special case
  double const chi = 1/x;
  double rho = (1+x)/kappa;                          // initial estimation
  for (size_t i = 0; i < imax; ++i) {
    double rhokappa = powk(rho, kappa-1);            // powk(x, k) = x^k, k integer
    if (abs2(1-rho*rhokappa*chi) < epsilon) break;
    rho = ((kappa-1)*rho + x/rhokappa)/kappa;        // next Heron iteration value
  }
  return rho;
}
```

Example, Modular C

Heron's method for the approximation of $\sqrt[kappa]{x}$

```
#pragma CMOD module phi    = my_proj::heron
#pragma CMOD import math   = C::math

double phi(double x, unsigned kappa) {
  if (kappa == 1) return x;
  double const chi = 1/x;
  double rho = (1+x)/kappa;
  for (size_t i = 0; i < imax; ++i) {
    double rhokappa = math::powk(rho, kappa-1);
    if (math::abs2(1-rho*rhokappa*chi) < epsilon) break;
    rho = ((kappa-1)*rho + x/rhokappa)/kappa;
  }
  return rho;
}
```

Example, Modular C, snippet

generic method (real or complex) for one branch of $\sqrt[\kappa]{x}$

```
#pragma CMOD module my_proj::snippet::heron

#pragma CMOD snippet none
#pragma CMOD slot RC  = complete
#pragma CMOD slot φ   = extern RC φ(RC x, unsigned κ);

RC φ(RC x, unsigned κ) {                    // replace greek names by greek symbols
  if (κ == 1) return x;
  RC const χ = 1/x;
  RC ρ = (1+x)/κ;
  for (C::size i = 0; i < imax; ++i) {
    RC ρκ = powk(ρ, κ-1);                   // powk is a type generic function
    if (abs2(1-ρ*ρκ*χ) < ε) break;          // abs2 is a type generic function
    ρ = ((κ-1)*ρ + x/ρκ)/κ;
  }
  return ρ;
}
```

Example, Modular C, snippet usage

generic method (real or complex) for one branch of $\sqrt[\kappa]{x}$

```
#pragma CMOD module my_proj::sqrt

#pragma CMOD import heronf    = my_proj::snippet::heron
#pragma CMOD fill   heronf::RC = C::real::float
#pragma CMOD fill   heronf::φ  = my_proj::sqrtf

#pragma CMOD import herond    = my_proj::snippet::heron
#pragma CMOD fill   herond::RC = C::real::double
#pragma CMOD fill   herond::φ  = my_proj::sqrt

#pragma CMOD import heronc    = my_proj::snippet::heron
#pragma CMOD fill   heronc::RC = C::complex::double
#pragma CMOD fill   heronc::φ  = my_proj::csqrt
```

Example, arbogast

generic method for $\sqrt[\kappa]{x}$ or an estimation of $\sqrt[\kappa]{x_0 + x_1 t + x_2 t^2 + \cdots}$

```
#pragma CMOD module          my_proj::snippet::heron
#pragma CMOD context DA = ⟪ ⟫

#pragma CMOD snippet none
#pragma slot TRC = complete
#pragma slot φ = extern RC φ(RC x, unsigned κ);

TRC φ(TRC x, unsigned κ) {
  if (κ == 1) return x;
  TRC const χ = ⟪1/x⟫;                    // may be polynomial division
  TRC ρ = ⟪(1+x)/κ⟫;                      // may be division of polynomial by integer
  for (C::size i = 0; i < imax; ++i) {  // for loop uses conventional operations
    TRC ρκ = powk(ρ, κ-1);                // powk is type generic on TRC
    if ( ⟪abs2(1-ρ*ρκ*χ) < ε⟫ ) break; // abs2 returns real type
    ρ = ⟪((κ-1)*ρ + x/ρκ)/κ⟫;
  }
  return ρ;
}
```

Table of Contents

arbogast, a Modular C package

arbogast differential operator I

```
#pragma CMOD module arbogast::snippet::ODE2nd
#pragma CMOD import arbogast::symbols

#pragma CMOD snippet none
/* The Taylor and floating types used   */
/* in the snippet                       */
#pragma CMOD slot TP = complete
#pragma CMOD slot Tα = complete
#pragma CMOD slot Tβ = complete
#pragma CMOD slot Tγ = complete
#pragma CMOD slot Tf = complete
```

arbogast, a Modular C package

arbogast differential operator II

```
/* The five functional parameters */
#pragma CMOD slot φ⁰ = { Tf a, b; b = φ⁰(a); }
#pragma CMOD slot φ¹ = { Tf a, b; b = φ¹(a); }
#pragma CMOD slot α  = none
#pragma CMOD slot β  = none
#pragma CMOD slot γ  = none
/* The name of the resulting function */
#pragma CMOD slot ¯  = extern TP ¯(TP);
```

arbogast, a Modular C package

arbogast differential operator III

```
/* Generic code of DO¯ */
TP ¯(TP z) {
  // Initialization
  TP z¹ = deriv(z);
  TP z² = deriv(z¹);
  // Mathematical function φ
  Tf r⁰ = φ⁰(z.coeff[0]);
  Tf r¹ = φ¹(z.coeff[0]);
  TP v⁰ = INITIALIZER(v⁰, -1, [0] = r⁰, [1] = r¹*z.coeff[1]);
  Tα const A    = arbogast::operate(α, z);
  Tβ const B    = arbogast::operate(β, z);
  Tγ const `    = arbogast::operate(γ, z);
  // Auxiliary variables
  TP const γα     = ⟪ -(`/A)*(z¹*z¹) ⟫;
  TP const βα     = ⟪ z²/z¹-(B/A)*z¹ ⟫;
  /* resolve by equating coefficients */
  ...
  return v⁰;
}
```

arbogast, Conclusion

arbogast in summary

- arbogast implements HOAD based on Modular C
  - modular implementation
  - modular usage
- extendable to new operators and special functions
- type generic interfaces based on C11's _Generic
- explicit instantiation
- it guarantees a complexity of $O(N^2)$ for all its operators
- it is fast
  - at compilation
  - at execution
- it is open source
- it is fun

Conclusion

Thanks for your invaluable contribution

Conclusion

Thanks for your attention