Validated integration of variational equations for ODEs and applications

Daniel Wilczak

Jagiellonian University, Kraków, Poland

Intl. Seminar on Interval Methods in Control Engineering

April 9, 2021, Kraków

Outline:

1. $\mathcal{C}^1$-algorithms for ODEs
2. Poincaré maps and derivatives
3. CAPD library: ODE solvers and Poincaré maps
4. Case study:
    - hyperbolic periodic orbits
    - solving BVP
    - hyperbolic chaos in the Rössler system
    - connecting orbits between periodic orbits
5. Applications:
    - hyperbolic chaotic attractor in the Kuznetsov system
    - chaos and connecting orbits in infinite dimension

Outline:

1. $\mathcal{C}^1$-algorithms for ODEs
2. Poincaré maps and derivatives
3. CAPD library: ODE solvers and Poincaré maps
4. Case study:
   - hyperbolic periodic orbits
   - solving BVP
   - hyperbolic chaos in the Rössler system
   - connecting orbits between periodic orbits
5. Applications:
   - hyperbolic chaotic attractor in the Kuznetsov system
   - chaos and connecting orbits in infinite dimension

Outline:

1. $\mathcal{C}^1$-algorithms for ODEs
2. Poincaré maps and derivatives
3. CAPD library: ODE solvers and Poincaré maps
4. Case study:
    - hyperbolic periodic orbits
    - solving BVP
    - hyperbolic chaos in the Rössler system
    - connecting orbits between periodic orbits
5. Applications:
    - hyperbolic chaotic attractor in the Kuznetsov system
    - chaos and connecting orbits in infinite dimension

$\mathcal{C}^1$ Solvers

Variational equation

Problem to solve:

$$\begin{aligned} x'(t) &= f(x(t)) && - \quad \text{main ODE} \\ V'(t) &= Df(x(t)) \cdot V(t) && - \quad \text{variational equation} \end{aligned}$$

or:

$$V(t) = D_x\varphi(t, x)$$

Initial conditions:

$$\begin{aligned} x(0) &\in [X] \\ V(0) &\in [V] \qquad \text{often } [V] = \{\mathrm{Id}\} \end{aligned}$$

Variational equation

Problem to solve:

$$\begin{aligned} x'(t) &= f(x(t)) & - & \quad \text{main ODE} \\ V'(t) &= Df(x(t)) \cdot V(t) & - & \quad \text{variational equation} \end{aligned}$$

or:

$$V(t) \;=\; D_x\varphi(t,x)$$

Initial conditions:

$$\begin{aligned} x(0) &\in [X] \\ V(0) &\in [V] \qquad \text{often } [V] = \{\mathrm{Id}\} \end{aligned}$$

Variational equation

Problem to solve:

$$\begin{aligned} x'(t) &= f(x(t)) && - \quad \text{main ODE} \\ V'(t) &= Df(x(t)) \cdot V(t) && - \quad \text{variational equation} \end{aligned}$$

or:

$$V(t) \;=\; D_x\varphi(t, x)$$

Initial conditions:

$$\begin{aligned} x(0) &\in [X] \\ V(0) &\in [V] \qquad \text{often } [V] = \{\mathrm{Id}\} \end{aligned}$$

Example (van der Pol oscillator)

$$x' = y, \quad y' = (1 - x^2)y - x$$

Full $\mathcal{C}^1$ system

$$\begin{cases} x' & = y \\ y' & = (1 - x^2)y - x \\ V'_{11} & = V_{21} \\ V'_{12} & = V_{22} \\ V'_{21} & = V_{21}(1 - x^2) - V_{11}(1 + 2xy) \\ V'_{22} & = V_{22}(1 - x^2) - V_{12}(1 + 2xy) \end{cases}$$

One can apply any $\mathcal{C}^0$ solver to the above system but ...

Complexity problem

Solvers (at least VNODE, CAPD) are $O\left(\text{dimension}^3\right)$.
(to reduce wrapping effect)
For variational equations this gives $O((\text{dimension}^2)^3)$.

Example (van der Pol oscillator)

$$x' = y, \quad y' = (1 - x^2)y - x$$

Full $\mathcal{C}^1$ system

$$\begin{cases} x' &= y \\ y' &= (1 - x^2)y - x \\ V'_{11} &= V_{21} \\ V'_{12} &= V_{22} \\ V'_{21} &= V_{21}(1 - x^2) - V_{11}(1 + 2xy) \\ V'_{22} &= V_{22}(1 - x^2) - V_{12}(1 + 2xy) \end{cases}$$

One can apply any $\mathcal{C}^0$ solver to the above system but ...

Complexity problem

Solvers (at least VNODE, CAPD) are $O\left(\text{dimension}^3\right)$.

(to reduce wrapping effect)

For variational equations this gives $O((\text{dimension}^2)^3)$.

Example (van der Pol oscillator)

$$x' = y, \quad y' = (1 - x^2)y - x$$

Full $\mathcal{C}^1$ system

$$\begin{cases} x' &= y \\ y' &= (1 - x^2)y - x \\ V'_{11} &= V_{21} \\ V'_{12} &= V_{22} \\ V'_{21} &= V_{21}(1 - x^2) - V_{11}(1 + 2xy) \\ V'_{22} &= V_{22}(1 - x^2) - V_{12}(1 + 2xy) \end{cases}$$

One can apply any $\mathcal{C}^0$ solver to the above system but ...

Complexity problem

Solvers (at least VNODE, CAPD) are $O\left(\text{dimension}^3\right)$.

(to reduce wrapping effect)

For variational equations this gives $O((\text{dimension}^2)^3)$.

Example (van der Pol oscillator)

$$x' = y, \quad y' = (1 - x^2)y - x$$

Full $\mathcal{C}^1$ system

$$\begin{cases} x' &= y \\ y' &= (1 - x^2)y - x \\ V'_{11} &= V_{21} \\ V'_{12} &= V_{22} \\ V'_{21} &= V_{21}(1 - x^2) - V_{11}(1 + 2xy) \\ V'_{22} &= V_{22}(1 - x^2) - V_{12}(1 + 2xy) \end{cases}$$

One can apply any $\mathcal{C}^0$ solver to the above system but ...

Complexity problem

Solvers (at least VNODE, CAPD) are $O\left(\text{dimension}^3\right)$.

(to reduce wrapping effect)

For variational equations this gives $O((\text{dimension}^2)^3)$.

Structure of variational equation

Fact:

$$V'(t) = Df(x(t)) \cdot V(t)$$

is nonautonomous and linear in V. Thus

$$D_x\varphi(t+h,x) = D_x\varphi(h,\varphi(t,x))D_x\varphi(t,x)$$

It is enough to compute

$$D_x\varphi(h,[X])$$

where h is the time step.

Structure of variational equation

Fact:

$$V'(t) = Df(x(t)) \cdot V(t)$$

is nonautonomous and linear in V. Thus

$$D_x\varphi(t+h,x) = D_x\varphi(h,\varphi(t,x))D_x\varphi(t,x)$$

It is enough to compute

$$D_x\varphi(h,[X])$$

where h is the time step.

P. Zgliczyński, $\mathcal{C}^1$-Lohner algorithm, Found. Comp. Math, (2002), 2:429-465

Structure of variational equation

Fact:

$$V'(t) = Df(x(t)) \cdot V(t)$$

is nonautonomous and linear in V. Thus

$$D_x\varphi(t+h,x) = D_x\varphi(h,\varphi(t,x))D_x\varphi(t,x)$$

It is enough to compute

$$D_x\varphi(h,[X])$$

where h is the time step.

P. Zgliczyński, $\mathcal{C}^1$-Lohner algorithm, Found. Comp. Math, (2002), 2:429-465

Structure of variational equation

Φ – numerical method for ODE

$$\varphi(h,x) \in \Phi(h,x) + [R]$$

Wrapping effect reduced by

$$\varphi(h,[X]) \subset \Phi(h,x_0) + D_x\Phi(h,[X])([X]-x_0) + [R]$$

Key observation:

$$D_x\varphi(h,[X]) \subset D_x\Phi(h,[X]) + [R_V]$$

$D_x\Phi(h,[X])$ – is already computed in $\mathcal{C}^0$ step.

Additional cost:

- rough enclosure for variational part
- remainder R_V for variational part
- propagation of $V(t)$ (wrapping effect reduction for V)

Structure of variational equation

Φ – numerical method for ODE

$$\varphi(h,x) \in \Phi(h,x) + [R]$$

Wrapping effect reduced by

$$\varphi(h,[X]) \subset \Phi(h,x_0) + D_x\Phi(h,[X])([X]-x_0) + [R]$$

Key observation:

$$D_x\varphi(h,[X]) \subset D_x\Phi(h,[X]) + [R_V]$$

$D_x\Phi(h,[X])$ – is already computed in $\mathcal{C}^0$ step.

Additional cost:

- rough enclosure for variational part
- remainder R_V for variational part
- propagation of $V(t)$ (wrapping effect reduction for V)

Structure of variational equation

Φ – numerical method for ODE

$$\varphi(h,x) \in \Phi(h,x) + [R]$$

Wrapping effect reduced by

$$\varphi(h,[X]) \subset \Phi(h,x_0) + D_x\Phi(h,[X])([X]-x_0) + [R]$$

Key observation:

$$D_x\varphi(h,[X]) \subset D_x\Phi(h,[X]) + [R_V]$$

$D_x\Phi(h,[X])$ – is already computed in $\mathcal{C}^0$ step.

Additional cost:

- rough enclosure for variational part
- remainder R_V for variational part
- propagation of $V(t)$ (wrapping effect reduction for V)

Structure of variational equation

Φ – numerical method for ODE

$$\varphi(h, x) \in \Phi(h, x) + [R]$$

Wrapping effect reduced by

$$\varphi(h, [X]) \subset \Phi(h, x_0) + D_x\Phi(h, [X])([X] - x_0) + [R]$$

Key observation:

$$D_x\varphi(h, [X]) \subset D_x\Phi(h, [X]) + [R_V]$$

$D_x\Phi(h, [X])$ – is already computed in $\mathcal{C}^0$ step.

Additional cost:

- rough enclosure for variational part
- remainder R_V for variational part
- propagation of $V(t)$ (wrapping effect reduction for V)

Rough enclosure for variational equation

Fact:
If $\varphi(h, [X])$ exists then $D_x\varphi(h, [X])$ does, too.

Strategies:
$[Y]$ – enclosure for $\mathcal{C}^0$ part

- logarithmic norms
- FOE: find $[W]$ such that

$$\mathrm{Id} + [0,h]Df([Y]) \cdot [W] \subset [W]$$

- HOE: define

$$[W] = \sum_{i=0}^{r} [V]^{[r]}[0,h]^r + [R_V]$$

  and check

$$[W]^{[r+1]}[0,h]^{r+1} \subset [R_V]$$

G.F. Corliss, R. Rihm, Validating an A Priori Enclosure Using High-Order Taylor Series, In Scientific Computing, Computer Arithmetic, and Validated Numerics, (1996).

Rough enclosure for variational equation

Fact:

If $\varphi(h, [X])$ exists then $D_x\varphi(h, [X])$ does, too.

Strategies:

$[Y]$ – enclosure for $\mathcal{C}^0$ part

- logarithmic norms
- FOE: find $[W]$ such that

$$\mathrm{Id} + [0, h]Df([Y]) \cdot [W] \subset [W]$$

- HOE: define

$$[W] = \sum_{i=0}^{r} [V]^{[r]}[0, h]^r + [R_V]$$

and check

$$[W]^{[r+1]}[0, h]^{r+1} \subset [R_V]$$

G.F. Corliss, R. Rihm, Validating an A Priori Enclosure Using High-Order Taylor Series, In Scientific Computing, Computer Arithmetic, and Validated Numerics, (1996).

Rough enclosure for variational equation

Fact:
If $\varphi(h,[X])$ exists then $D_x\varphi(h,[X])$ does, too.

Strategies:
$[Y]$ – enclosure for $\mathcal{C}^0$ part

- logarithmic norms
- FOE: find $[W]$ such that

$$\mathrm{Id} + [0,h]Df([Y])\cdot[W] \subset [W]$$

- HOE: define

$$[W] = \sum_{i=0}^{r} [V]^{[r]}[0,h]^r + [R_V]$$

and check

$$[W]^{[r+1]}[0,h]^{r+1} \subset [R_V]$$

G.F. Corliss, R. Rihm, Validating an A Priori Enclosure Using High-Order Taylor Series, In Scientific Computing, Computer Arithmetic, and Validated Numerics, (1996).

Rough enclosure for variational equation

Fact:

If $\varphi(h,[X])$ exists then $D_x\varphi(h,[X])$ does, too.

Strategies:

$[Y]$ – enclosure for $\mathcal{C}^0$ part

- logarithmic norms
- FOE: find $[W]$ such that

$$\mathrm{Id} + [0,h]Df([Y])\cdot[W] \subset [W]$$

- HOE: define

$$[W] = \sum_{i=0}^{r}[V]^{[r]}[0,h]^r + [R_V]$$

and check

$$[W]^{[r+1]}[0,h]^{r+1} \subset [R_V]$$

G.F. Corliss, R. Rihm, Validating an A Priori Enclosure Using High-Order Taylor Series, In Scientific Computing, Computer Arithmetic, and Validated Numerics, (1996).

Propagation of matrices

Input:

$$\begin{aligned} D_x\varphi(h,\varphi(t,x)) &\in [A] \\ D_x\varphi(t,x) &\in X_0 + C[R_0] + B[R] \end{aligned}$$

Then

$$D_x\varphi(t+h,x) \in [A]X_0 + ([A]C)[R_0] + ([A]B)[R]$$

and use QR-like strategies to propagate products.

Facts

- worse control of the wrapping effect
  (dependency of V wrt to V is not used)
- much faster than direct application of a $\mathcal{C}^0$ algorithm

Propagation of matrices

Input:

$$\begin{aligned} D_x\varphi(h,\varphi(t,x)) &\in [A] \\ D_x\varphi(t,x) &\in X_0 + C[R_0] + B[R] \end{aligned}$$

Then

$$D_x\varphi(t+h,x) \in [A]X_0 + ([A]C)[R_0] + ([A]B)[R]$$

and use QR-like strategies to propagate products.

Facts

- worse control of the wrapping effect
  (dependency of V wrt to V is not used)
- much faster than direct application of a $\mathcal{C}^0$ algorithm

Propagation of matrices

Input:

$$\begin{aligned} D_x\varphi(h,\varphi(t,x)) &\in [A] \\ D_x\varphi(t,x) &\in X_0 + C[R_0] + B[R] \end{aligned}$$

Then

$$D_x\varphi(t+h,x) \in [A]X_0 + ([A]C)[R_0] + ([A]B)[R]$$

and use QR-like strategies to propagate products.

Facts

- worse control of the wrapping effect

  (dependency of V wrt to V is not used)
- much faster than direct application of a $\mathcal{C}^0$ algorithm

We have to enclose

$$D_x\varphi(h,\varphi(t,x)) \quad \in \quad [A]$$

Taylor method

P. Zgliczyński, $\mathcal{C}^1$-Lohner algorithm, Found. Comp. Math, (2002), 2:429-465

$$[A] = \sum_{i=0}^{r} [V]^{[r]} h^r + [R_V]$$

Hermite-Obreshkov method

N. S. Nedialkov, K. R. Jackson, 1998. Developments in Reliable Computing 5, 289–310.
I. Walawska, DW, Appl. Math. Comput., 291C (2016), 303-322

$$\Psi_{q,p}(h,V) \quad := \quad \sum_{k=0}^{q} \binom{p+q-k}{p} \Big/ \binom{p+q}{p} h^k V^{[k]}$$

Solve for $V(h) \in [A]$ using interval Krawczyk method

(The same matrices as for $\mathcal{C}^0$ part – no extra cost!)

$$\Psi_{q,p}(-h, V(h)) - \Psi_{p,q}(h, V(0)) \in [R_{VHO}]$$

We have to enclose

$$D_x\varphi(h,\varphi(t,x)) \quad \in \quad [A]$$

Taylor method

P. Zgliczyński, $\mathcal{C}^1$-Lohner algorithm, Found. Comp. Math, (2002), 2:429-465

$$[A] = \sum_{i=0}^{r} [V]^{[r]} h^r + [R_V]$$

Hermite-Obreshkov method

N. S. Nedialkov, K. R. Jackson, 1998. Developments in Reliable Computing 5, 289–310.
I. Walawska, DW, Appl. Math. Comput., 291C (2016), 303-322

$$\Psi_{q,p}(h,V) \quad := \quad \sum_{k=0}^{q} \binom{p+q-k}{p} / \binom{p+q}{p} h^k V^{[k]}$$

Solve for $V(h) \in [A]$ using interval Krawczyk method

(The same matrices as for $\mathcal{C}^0$ part – no extra cost!)

$$\Psi_{q,p}(-h,V(h)) - \Psi_{p,q}(h,V(0)) \in [R_{VHO}]$$

We have to enclose

$$D_x\varphi(h,\varphi(t,x)) \quad \in \quad [A]$$

Taylor method

P. Zgliczyński, $\mathcal{C}^1$-Lohner algorithm, Found. Comp. Math, (2002), 2:429-465

$$[A] = \sum_{i=0}^{r} [V]^{[r]} h^r + [R_V]$$

Hermite-Obreshkov method

N. S. Nedialkov, K. R. Jackson, 1998. Developments in Reliable Computing 5, 289–310.
I. Walawska, DW, Appl. Math. Comput., 291C (2016), 303-322

$$\Psi_{q,p}(h,V) \quad := \quad \sum_{k=0}^{q} \binom{p+q-k}{p} \Big/ \binom{p+q}{p} h^k V^{[k]}$$

Solve for $V(h) \in [A]$ using interval Krawczyk method

(The same matrices as for $\mathcal{C}^0$ part – no extra cost!)

$$\Psi_{q,p}(-h,V(h)) - \Psi_{p,q}(h,V(0)) \in [R_{VHO}]$$

We have to enclose

$$D_x\varphi(h,\varphi(t,x)) \quad \in \quad [A]$$

Taylor method

P. Zgliczyński, $\mathcal{C}^1$-Lohner algorithm, Found. Comp. Math, (2002), 2:429-465

$$[A] = \sum_{i=0}^{r}[V]^{[r]}h^r + [R_V]$$

Hermite-Obreshkov method

N. S. Nedialkov, K. R. Jackson, 1998. Developments in Reliable Computing 5, 289–310.
I. Walawska, DW, Appl. Math. Comput., 291C (2016), 303-322

$$\Psi_{q,p}(h, V) \quad := \quad \sum_{k=0}^{q}\binom{p+q-k}{p}\Big/\binom{p+q}{p}h^k V^{[k]}$$

Solve for $V(h) \in [A]$ using interval Krawczyk method

(The same matrices as for $\mathcal{C}^0$ part – no extra cost!)

$$\Psi_{q,p}(-h, V(h)) - \Psi_{p,q}(h, V(0)) \in [R_{VHO}]$$

Poincaré maps

Derivatives of Poincaré maps:

Section:

$$\Pi = \Pi_{\alpha,\mathcal{C}} = \{x : \alpha(\mathbf{x}) = \mathbf{0} \wedge \langle \nabla\alpha(x); f(x)\rangle \neq 0 \wedge \mathcal{C}(\mathbf{x})\}$$

Return time:

$$T : \mathbb{R}^n \to \mathbb{R}$$

Then

$$\alpha(\varphi(T(x), x)) \equiv 0$$
$$\Downarrow$$
$$\sum_{i=1}^{n} \frac{\partial\alpha}{\partial x_i}(P(x)) \left(f_i(P(x)) \frac{\partial T}{\partial x_j}(x) + \frac{\partial\varphi_i}{\partial x_j}(T(x), x) \right) = 0$$
$$\Downarrow$$
$$\frac{\partial T(x)}{\partial x_j} = -\frac{\langle \nabla\alpha(P(x))\,;\, D_{x_j}\varphi(T(x), x)\rangle}{\langle \nabla\alpha(P(x))\,;\, f(\varphi(T(x), x))\rangle}$$

Derivatives of Poincaré maps:

Section:

$$\Pi = \Pi_{\alpha,\mathcal{C}} = \{x : \alpha(\mathbf{x}) = \mathbf{0} \wedge \langle \nabla\alpha(x); f(x)\rangle \neq 0 \wedge \mathcal{C}(\mathbf{x})\}$$

Return time:

$$T : \mathbb{R}^n \to \mathbb{R}$$

Then

$$\alpha(\varphi(T(x), x)) \equiv 0$$

$$\Downarrow$$

$$\sum_{i=1}^{n} \frac{\partial\alpha}{\partial x_i}(P(x)) \left(f_i(P(x)) \frac{\partial T}{\partial x_j}(x) + \frac{\partial\varphi_i}{\partial x_j}(T(x), x) \right) = 0$$

$$\Downarrow$$

$$\frac{\partial T(x)}{\partial x_j} = -\frac{\langle \nabla\alpha(P(x))\,;\, D_{x_j}\varphi(T(x), x)\rangle}{\langle \nabla\alpha(P(x))\,;\, f(\varphi(T(x), x))\rangle}$$

Derivatives of Poincaré maps:

Section:

$$\Pi = \Pi_{\alpha,\mathcal{C}} = \{x : \alpha(\mathbf{x}) = \mathbf{0} \wedge \langle \nabla\alpha(x); f(x)\rangle \neq 0 \wedge \mathcal{C}(\mathbf{x})\}$$

Return time:

$$T : \mathbb{R}^n \to \mathbb{R}$$

Then

$$\alpha(\varphi(T(x), x)) \equiv 0$$

$$\Downarrow$$

$$\sum_{i=1}^{n} \frac{\partial\alpha}{\partial x_i}(P(x)) \left(f_i(P(x)) \frac{\partial T}{\partial x_j}(x) + \frac{\partial\varphi_i}{\partial x_j}(T(x), x) \right) = 0$$

$$\Downarrow$$

$$\frac{\partial T(x)}{\partial x_j} = -\frac{\langle \nabla\alpha(P(x))\,;\, D_{x_j}\varphi(T(x), x)\rangle}{\langle \nabla\alpha(P(x))\,;\, f(\varphi(T(x), x))\rangle}$$

Derivatives of Poincaré maps:

Section:

$$\Pi = \Pi_{\alpha,\mathcal{C}} = \{x : \alpha(\mathbf{x}) = \mathbf{0} \wedge \langle \nabla\alpha(x); f(x)\rangle \neq 0 \wedge \mathcal{C}(\mathbf{x})\}$$

Return time:

$$T : \mathbb{R}^n \to \mathbb{R}$$

Then

$$\alpha(\varphi(T(x), x)) \equiv 0$$
$$\Downarrow$$
$$\sum_{i=1}^{n} \frac{\partial\alpha}{\partial x_i}(P(x)) \left(f_i(P(x)) \frac{\partial T}{\partial x_j}(x) + \frac{\partial\varphi_i}{\partial x_j}(T(x), x) \right) = 0$$
$$\Downarrow$$
$$\frac{\partial T(x)}{\partial x_j} = -\frac{\langle \nabla\alpha(P(x))\,;\, D_{x_j}\varphi(T(x), x)\rangle}{\langle \nabla\alpha(P(x))\,;\, f(\varphi(T(x), x))\rangle}$$

Derivatives of Poincaré maps:

Section:

$$\Pi = \Pi_{\alpha,\mathcal{C}} = \{x : \alpha(\mathbf{x}) = \mathbf{0} \wedge \langle \nabla\alpha(x); f(x)\rangle \neq 0 \wedge \mathcal{C}(\mathbf{x})\}$$

Return time:

$$T : \mathbb{R}^n \to \mathbb{R}$$

Then

$$\alpha(\varphi(T(x), x)) \equiv 0$$

$$\Downarrow$$

$$\sum_{i=1}^{n} \frac{\partial\alpha}{\partial x_i}(P(x)) \left(f_i(P(x)) \frac{\partial T}{\partial x_j}(x) + \frac{\partial \varphi_i}{\partial x_j}(T(x), x) \right) = 0$$

$$\Downarrow$$

$$\frac{\partial T(x)}{\partial x_j} = -\frac{\langle \nabla\alpha(P(x))\,;\, D_{x_j}\varphi(T(x), x)\rangle}{\langle \nabla\alpha(P(x))\,;\, f(\varphi(T(x), x))\rangle}$$

Derivatives of Poincaré maps:

Given bounds for:

- $T([X])$ – from $\mathcal{C}^0$ part
- $P([X])$ – from $\mathcal{C}^0$ part
- $D_x\varphi(T([X]),[X])$

Compute:

- derivatives for return time

$$\frac{\partial T}{\partial x_j}(x) = -\frac{\langle \nabla\alpha(P(x))\,;\, D_{x_j}\varphi(T(x),x)\rangle}{\langle \nabla\alpha(P(x))\,;\, f(\varphi(T(x),x))\rangle}$$

- derivatives of Poincaré map

$$\frac{\partial P_i}{\partial x_j}(x) = \frac{\partial \varphi_i}{\partial x_j}(T(x),x) + \sum_{k=1}^{n} f_i(P(x))\frac{\partial T}{\partial x_j}(x)$$

Derivatives of Poincaré maps:

Given bounds for:

- $T([X])$ – from $\mathcal{C}^0$ part
- $P([X])$ – from $\mathcal{C}^0$ part
- $D_x\varphi(T([X]),[X])$

Compute:

- derivatives for return time

$$\frac{\partial T}{\partial x_j}(x) = -\frac{\langle \nabla\alpha(P(x))\,;\,D_{x_j}\varphi(T(x),x)\rangle}{\langle \nabla\alpha(P(x))\,;\,f(\varphi(T(x),x))\rangle}$$

- derivatives of Poincaré map

$$\frac{\partial P_i}{\partial x_j}(x) = \frac{\partial \varphi_i}{\partial x_j}(T(x),x) + \sum_{k=1}^{n} f_i(P(x))\frac{\partial T}{\partial x_j}(x)$$

Derivatives of Poincaré maps:

Given bounds for:

- $T([X])$ – from $\mathcal{C}^0$ part
- $P([X])$ – from $\mathcal{C}^0$ part
- $D_x\varphi(T([X]),[X])$

Compute:

- derivatives for return time

$$\frac{\partial T}{\partial x_j}(x) = -\frac{\langle \nabla\alpha(P(x))\,;\, D_{x_j}\varphi(T(x),x)\rangle}{\langle \nabla\alpha(P(x))\,;\, f(\varphi(T(x),x))\rangle}$$

- derivatives of Poincaré map

$$\frac{\partial P_i}{\partial x_j}(x) = \frac{\partial \varphi_i}{\partial x_j}(T(x),x) + \sum_{k=1}^{n} f_i(P(x))\frac{\partial T}{\partial x_j}(x)$$

Derivatives of Poincaré maps:

Given bounds for:

- $T([X])$ – from $\mathcal{C}^0$ part
- $P([X])$ – from $\mathcal{C}^0$ part
- $D_x\varphi(T([X]),[X])$

Compute:

- derivatives for return time

$$\frac{\partial T}{\partial x_j}(x) = -\frac{\left\langle \nabla\alpha(P(x))\,;\; D_{x_j}\varphi(T(x),x)\right\rangle}{\left\langle \nabla\alpha(P(x))\,;\; f(\varphi(T(x),x))\right\rangle}$$

- derivatives of Poincaré map

$$\frac{\partial P_i}{\partial x_j}(x) = \frac{\partial \varphi_i}{\partial x_j}(T(x),x) + \sum_{k=1}^{n} f_i(P(x))\frac{\partial T}{\partial x_j}(x)$$

Derivatives of Poincaré maps:

Given bounds for:

- $T([X])$ – from $\mathcal{C}^0$ part
- $P([X])$ – from $\mathcal{C}^0$ part
- $D_x\varphi(T([X]),[X])$

Compute:

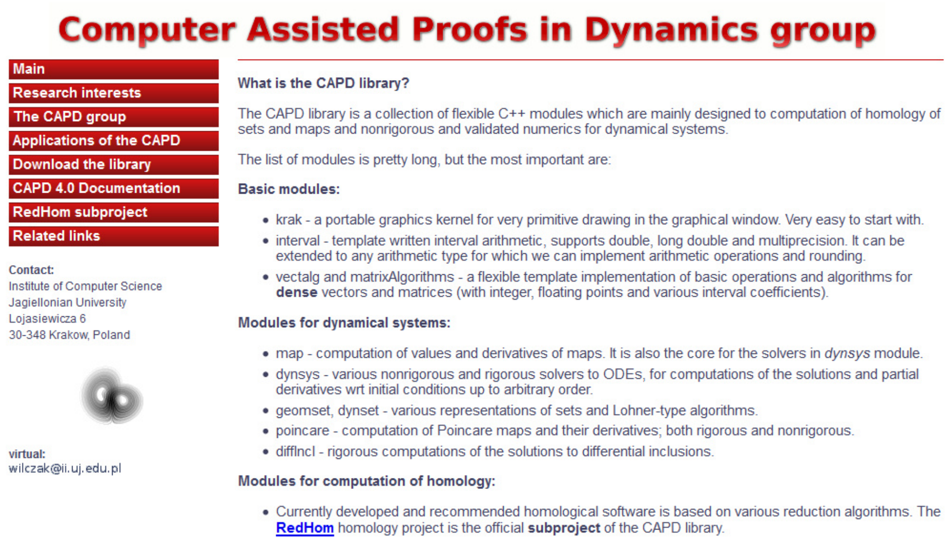
- derivatives for return time

$$\frac{\partial T}{\partial x_j}(x) = -\frac{\left\langle \nabla\alpha(P(x))\,;\; D_{x_j}\varphi(T(x),x)\right\rangle}{\left\langle \nabla\alpha(P(x))\,;\; f(\varphi(T(x),x))\right\rangle}$$

- derivatives of Poincaré map

$$\frac{\partial P_i}{\partial x_j}(x) = \frac{\partial \varphi_i}{\partial x_j}(T(x),x) + \sum_{k=1}^{n} f_i(P(x))\frac{\partial T}{\partial x_j}(x)$$

CAPD library



Computer Assisted Proofs in Dynamics group

Contact:
Institute of Computer Science
Jagiellonian University
Lojasiewicza 6
30-348 Krakow, Poland

virtual:
wilczak@ii.uj.edu.pl

What is the CAPD library?

The CAPD library is a collection of flexible C++ modules which are mainly designed to computation of homology of sets and maps and nonrigorous and validated numerics for dynamical systems.

The list of modules is pretty long, but the most important are:

Basic modules:

- krak - a portable graphics kernel for very primitive drawing in the graphical window. Very easy to start with.
- interval - template written interval arithmetic, supports double, long double and multiprecision. It can be extended to any arithmetic type for which we can implement arithmetic operations and rounding.
- vectalg and matrixAlgorithms - a flexible template implementation of basic operations and algorithms for **dense** vectors and matrices (with integer, floating points and various interval coefficients).

Modules for dynamical systems:

- map - computation of values and derivatives of maps. It is also the core for the solvers in *dynsys* module.
- dynsys - various nonrigorous and rigorous solvers to ODEs, for computations of the solutions and partial derivatives wrt initial conditions up to arbitrary order.
- geomset, dynset - various representations of sets and Lohner-type algorithms.
- poincare - computation of Poincare maps and their derivatives; both rigorous and nonrigorous.
- diffIncl - rigorous computations of the solutions to differential inclusions.

Modules for computation of homology:

- Currently developed and recommended homological software is based on various reduction algorithms. The RedHom homology project is the official **subproject** of the CAPD library.

http://capd.ii.uj.edu.pl

Computer Assisted Proofs in Dynamics

1992 Marian Mrozek

First version of rigorous ODE solver.

1995 M.Mrozek, K.Mischaikow

Encyclopedia Britanica:

paper on chaos in the Lorenz equations amog 4 best results in mathematics in 1995

2001

CAPD publically available at **http://capd.ii.uj.edu.pl**

1992-2021

continuously developed at the Jagiellonian University

1992 Marian Mrozek

First version of rigorous ODE solver.

1995 M.Mrozek, K.Mischaikow

Encyclopedia Britanica:
paper on chaos in the Lorenz equations amog 4 best results in mathematics in 1995

2001

CAPD publically available at **http://capd.ii.uj.edu.pl**

1992-2021

continuously developed at the Jagiellonian University

1992 Marian Mrozek

First version of rigorous ODE solver.

1995 M.Mrozek, K.Mischaikow

Encyclopedia Britanica:

paper on chaos in the Lorenz equations amog 4 best results in mathematics in 1995

2001

CAPD publically available at **http://capd.ii.uj.edu.pl**

1992-2021

continuously developed at the Jagiellonian University

The CAPD in 2021:

- core CAPD: (Multiprecision) IA, linear algebra (dense)
- **CAPD::RedHom: (co)-homology software**

Paweł Pilarczyk   Paweł Dłotko   Mateusz Juda

- **CAPD::DynSys: validated numerics for dynamics**

Piotr Zgliczyński   Tomasz Kapela   Daniel Wilczak

The CAPD::DynSys in 2021

- $\mathcal{C}^0 - \mathcal{C}^1 - \mathcal{C}^r$ ODE solvers
- Poincaré maps and their $r-$th order derivatives
- Differential inclusions
- dissipative PDEs (C^0 and C^1 algorithms)
- supports: double, long double, multiprecision, interval, mpfr-intervals

Some applications:

- $\mathcal{C}^0$-computations;

  chaotic dynamics for many textbook systems, bifurcations for reversible systems
- $\mathcal{C}^1$-computations;

  periodic orbits (in quite high dimensions, like 300 for the N-body problem), hyperbolicity, connecting orbits for ODEs both to equilibria and periodic solutions
- $\mathcal{C}^2 - \mathcal{C}^5$ computations;

  global cocoon bifurcations, homoclinic tangencies, bifurcations of periodic orbits for ODEs, KAM stability of elliptic periodic solutions, invariant tori
- PDEs;

  chaos and connecting orbits

The CAPD::DynSys in 2021

- $\mathcal{C}^0 - \mathcal{C}^1 - \mathcal{C}^r$ ODE solvers
- Poincaré maps and their $r-$th order derivatives
- Differential inclusions
- dissipative PDEs (C^0 and C^1 algorithms)
- supports: double, long double, multiprecision, interval, mpfr-intervals

Some applications:

- $\mathcal{C}^0$-computations;

  chaotic dynamics for many textbook systems, bifurcations for reversible systems
- $\mathcal{C}^1$-computations;

  periodic orbits (in quite high dimensions, like 300 for the N-body problem), hyperbolicity, connecting orbits for ODEs both to equilibria and periodic solutions
- $\mathcal{C}^2 - \mathcal{C}^5$ computations;

  global cocoon bifurcations, homoclinic tangencies, bifurcations of periodic orbits for ODEs, KAM stability of elliptic periodic solutions, invariant tori
- PDEs;

  chaos and connecting orbits

The CAPD::DynSys in 2021

- $\mathcal{C}^0 - \mathcal{C}^1 - \mathcal{C}^r$ ODE solvers
- Poincaré maps and their $r-$th order derivatives
- Differential inclusions
- dissipative PDEs (C^0 and C^1 algorithms)
- supports: double, long double, multiprecision, interval, mpfr-intervals

Some applications:

- $\mathcal{C}^0$-computations;

  chaotic dynamics for many textbook systems, bifurcations for reversible systems
- $\mathcal{C}^1$-computations;

  periodic orbits (in quite high dimensions, like 300 for the N-body problem), hyperbolicity, connecting orbits for ODEs both to equilibria and periodic solutions
- $\mathcal{C}^2 - \mathcal{C}^5$ computations;

  global cocoon bifurcations, homoclinic tangencies, bifurcations of periodic orbits for ODEs, KAM stability of elliptic periodic solutions, invariant tori
- PDEs;

  chaos and connecting orbits

The CAPD::DynSys in 2021

- $\mathcal{C}^0 - \mathcal{C}^1 - \mathcal{C}^r$ ODE solvers
- Poincaré maps and their $r-$th order derivatives
- Differential inclusions
- dissipative PDEs (C^0 and C^1 algorithms)
- supports: double, long double, multiprecision, interval, mpfr-intervals

Some applications:

- $\mathcal{C}^0$-computations;

  chaotic dynamics for many textbook systems, bifurcations for reversible systems
- $\mathcal{C}^1$-computations;

  periodic orbits (in quite high dimensions, like 300 for the N-body problem), hyperbolicity, connecting orbits for ODEs both to equilibria and periodic solutions
- $\mathcal{C}^2 - \mathcal{C}^5$ computations;

  global cocoon bifurcations, homoclinic tangencies, bifurcations of periodic orbits for ODEs, KAM stability of elliptic periodic solutions, invariant tori
- PDEs;

  chaos and connecting orbits

The CAPD::DynSys in 2021

- $\mathcal{C}^0 - \mathcal{C}^1 - \mathcal{C}^r$ ODE solvers
- Poincaré maps and their $r-$th order derivatives
- Differential inclusions
- dissipative PDEs (C^0 and C^1 algorithms)
- supports: double, long double, multiprecision, interval, mpfr-intervals

Some applications:

- $\mathcal{C}^0$-computations;

  chaotic dynamics for many textbook systems, bifurcations for reversible systems
- $\mathcal{C}^1$-computations;

  periodic orbits (in quite high dimensions, like 300 for the N-body problem), hyperbolicity, connecting orbits for ODEs both to equilibria and periodic solutions
- $\mathcal{C}^2 - \mathcal{C}^5$ computations;

  global cocoon bifurcations, homoclinic tangencies, bifurcations of periodic orbits for ODEs, KAM stability of elliptic periodic solutions, invariant tori
- PDEs;

  chaos and connecting orbits

The CAPD::DynSys in 2021

- $\mathcal{C}^0 - \mathcal{C}^1 - \mathcal{C}^r$ ODE solvers
- Poincaré maps and their $r-$th order derivatives
- Differential inclusions
- dissipative PDEs (C^0 and C^1 algorithms)
- supports: double, long double, multiprecision, interval, mpfr-intervals

Some applications:

- $\mathcal{C}^0$-computations;

  chaotic dynamics for many textbook systems, bifurcations for reversible systems
- $\mathcal{C}^1$-computations;

  periodic orbits (in quite high dimensions, like 300 for the N-body problem), hyperbolicity, connecting orbits for ODEs both to equilibria and periodic solutions
- $\mathcal{C}^2 - \mathcal{C}^5$ computations;

  global cocoon bifurcations, homoclinic tangencies, bifurcations of periodic orbits for ODEs, KAM stability of elliptic periodic solutions, invariant tori
- PDEs;

  chaos and connecting orbits

Compilation of the library:

- `./configure [options]`
- `make -j`

  (takes some time, more than 120 000 lines of code)

Basic options:

- `--with-mpfr`
- `--with-wx-config` (internal graphics kernel)

Building own programs:

```
g++ main.cpp -o main `capd-config --cflags --libs`
```

Optional scripts:

```
`capd-gui-config --cflags --libs`
`mpcapd-config --cflags --libs`
```

Compilation of the library:

- `./configure [options]`
- `make -j`

  (takes some time, more than 120 000 lines of code)

Basic options:

- `--with-mpfr`
- `--with-wx-config` (internal graphics kernel)

Building own programs:

```
g++ main.cpp -o main `capd-config --cflags --libs`
```

Optional scripts:

```
`capd-gui-config --cflags --libs`
`mpcapd-config --cflags --libs`
```

Compilation of the library:

- `./configure [options]`
- `make -j`

  (takes some time, more than 120 000 lines of code)

Basic options:

- `--with-mpfr`
- `--with-wx-config` (internal graphics kernel)

Building own programs:

```
g++ main.cpp -o main `capd-config --cflags --libs`
```

Optional scripts:

```
`capd-gui-config --cflags --libs`
`mpcapd-config --cflags --libs`
```

Compilation of the library:

- `./configure [options]`
- `make -j`

  (takes some time, more than 120 000 lines of code)

Basic options:

- `--with-mpfr`
- `--with-wx-config` (internal graphics kernel)

Building own programs:

```
g++ main.cpp -o main `capd-config --cflags --libs`
```

Optional scripts:

```
`capd-gui-config --cflags --libs`
`mpcapd-config --cflags --libs`
```

Main header files

```
#include "capd/capdlib.h"
#include "capd/mpcapdlib.h" // for multiprecision
```

Defined types:

- interval, MpFloat, MpInterval
- **Algebra:**
  - **D**Vector, **LD**Vector, **I**Vector, **Mp**Vector, **MpI**Vector
  - **[Prefix]**Matrix
  - **[Prefix]**Hessian
  - **[Prefix]**Jet
- **Automatic differentiation:**
  - **[Prefix]**Map
- **ODE solvers:**
  - **[Prefix]**OdeSolver, **[Prefix]**CnOdeSolver
  - **[Prefix]**TimeMap, **[Prefix]**CnTimeMap
  - **[Prefix]**PoincareMap, **[Prefix]**CnPoincareMap

Main header files

```
#include "capd/capdlib.h"
#include "capd/mpcapdlib.h" // for multiprecision
```

Defined types:

- interval, MpFloat, MpInterval
- **Algebra:**
  - **D**Vector, **LD**Vector, **I**Vector, **Mp**Vector, **MpI**Vector
  - **[Prefix]**Matrix
  - **[Prefix]**Hessian
  - **[Prefix]**Jet
- **Automatic differentiation:**
  - **[Prefix]**Map
- **ODE solvers:**
  - **[Prefix]**OdeSolver, **[Prefix]**CnOdeSolver
  - **[Prefix]**TimeMap, **[Prefix]**CnTimeMap
  - **[Prefix]**PoincareMap, **[Prefix]**CnPoincareMap

Main header files

```
#include "capd/capdlib.h"
#include "capd/mpcapdlib.h" // for multiprecision
```

Defined types:

- interval, MpFloat, MpInterval
- **Algebra:**
  - **D**Vector, **LD**Vector, **I**Vector, **Mp**Vector, **MpI**Vector
  - **[Prefix]**Matrix
  - **[Prefix]**Hessian
  - **[Prefix]**Jet
- **Automatic differentiation:**
  - **[Prefix]**Map
- **ODE solvers:**
  - **[Prefix]**OdeSolver, **[Prefix]**CnOdeSolver
  - **[Prefix]**TimeMap, **[Prefix]**CnTimeMap
  - **[Prefix]**PoincareMap, **[Prefix]**CnPoincareMap

Main header files

```
#include "capd/capdlib.h"
#include "capd/mpcapdlib.h" // for multiprecision
```

Defined types:

- interval, MpFloat, MpInterval
- **Algebra:**
  - **D**Vector, **LD**Vector, **I**Vector, **Mp**Vector, **MpI**Vector
  - **[Prefix]**Matrix
  - **[Prefix]**Hessian
  - **[Prefix]**Jet
- **Automatic differentiation:**
  - **[Prefix]**Map
- **ODE solvers:**
  - **[Prefix]**OdeSolver, **[Prefix]**CnOdeSolver
  - **[Prefix]**TimeMap, **[Prefix]**CnTimeMap
  - **[Prefix]**PoincareMap, **[Prefix]**CnPoincareMap

Main header files

```
#include "capd/capdlib.h"
#include "capd/mpcapdlib.h" // for multiprecision
```

Defined types:

- interval, MpFloat, MpInterval
- **Algebra:**
  - **D**Vector, **LD**Vector, **I**Vector, **Mp**Vector, **MpI**Vector
  - **[Prefix]**Matrix
  - **[Prefix]**Hessian
  - **[Prefix]**Jet
- **Automatic differentiation:**
  - **[Prefix]**Map
- **ODE solvers:**
  - **[Prefix]**OdeSolver, **[Prefix]**CnOdeSolver
  - **[Prefix]**TimeMap, **[Prefix]**CnTimeMap
  - **[Prefix]**PoincareMap, **[Prefix]**CnPoincareMap

Main header files

```
#include "capd/capdlib.h"
#include "capd/mpcapdlib.h" // for multiprecision
```

Defined types:

- interval, MpFloat, MpInterval
- **Algebra:**
  - **D**Vector, **LD**Vector, **I**Vector, **Mp**Vector, **MpI**Vector
  - **[Prefix]**Matrix
  - **[Prefix]**Hessian
  - **[Prefix]**Jet
- **Automatic differentiation:**
  - **[Prefix]**Map
- **ODE solvers:**
  - **[Prefix]**OdeSolver, **[Prefix]**CnOdeSolver
  - **[Prefix]**TimeMap, **[Prefix]**CnTimeMap
  - **[Prefix]**PoincareMap, **[Prefix]**CnPoincareMap

```
// Note: all examples in this tutorial require C++11 support
#include <iostream>
#include "capd/capdlib.h"
using namespace capd;
using namespace std;
int main(){
  IMap f("par:a;var:x,y;fun:sin(x*cos(y)),y*cos(a*y)-x^2;",3);
  f.setParameter("a",interval(1.,1.01));
  IVector u({1,3});
  cout << "f(u)=" << f(u) << endl;
  cout << "Df(u)=" << f.derivative(u) << endl;
  IJet jet(2,3);  // third order Taylor coefficients, two variables
  f(u,jet);       // compute full Taylor expansion
  cout << jet.toString() << endl;

  IOdeSolver solver(f,20);  // ODE integrator
  ITimeMap tm(solver);      // class for long time integration
  C0HOTripletonSet set(u);  // representation of initial condition
  // integrate until T=2 and print result
  cout << "phi(2,(1,3)) = " << tm(2.,set) << endl;

  ITimeMap::SolutionCurve solution(0.);
  set = C0HOTripletonSet(IVector({1,1}));
  // integrate until T=2 and save entire trajectory
  tm(2.,set,solution);
  cout << "solution(2)=" << solution(2) << endl;
  cout << "solution(0.9,1.1)=" << solution(interval(.9,1.1)) << endl;
}
```

```
// Note: all examples in this tutorial require C++11 support
#include <iostream>
#include "capd/capdlib.h"
#include "capd/mpcapdlib.h"
using namespace capd;
using namespace std;
int main(){
  cout.precision(60);
  MpFloat::setDefaultPrecision(200);
  MpIMap lorenz("var:x,y,z;fun:10*(y-x),x*(28-z)-y,x*y-8*z/3;");

  MpIOdeSolver solver(lorenz,40);  // ODE integrator
  MpITimeMap tm(solver);           // class for long time integration
  // initial condition
  MpIVector u({MpInterval(1.),MpInterval(3.),MpInterval(10.)});
  // tripleton representation of initial condition
  MpC0TripletonSet set(u);
  // integrate until T=2 and print result
  cout << "phi(2,(1,3,10)) = " << tm(MpInterval(2.),set) << endl;
}
/* Output (reformatted to fit presentation window)
 phi(2,(1,3,10)) = {
 [-2.17336885511749012718999487044194479578491574751054376072653 ,
  -2.17336885511749012718999487044194479578491574751054376063012 ],
 [-1.81501617053155848885754667427341477756244393385646228596O357 ,
  -1.81501617053155848885754667427341477756244393385646228594609 2 ],
 [2.04129457433159136349537343394345779855485364177570327972290e1 ,
  2.04129457433159136349537343394345779855485364177570327973768e1 ]}*/
```

```
/** Example of integration of variational equations **/
#include <iostream>
#include "capd/capdlib.h"
using namespace capd;
using namespace std;
int main(){
  IMap lorenz("var:x,y,z;fun:10*(y-x),x*(28-z)-y,x*y-8*z/3;");
  IOdeSolver solver(lorenz,20);  // ODE integrator
  ITimeMap tm(solver);           // class for long time integration

  IVector u({1,5,23});
  // representation of initial condition,
  // initial condition for variational equations is Id by default
  C1HORect2Set set(u);
  // integrate until T=2 and print result
  cout << "phi(2,u)=" << tm(2.,set) << endl;
  // print monodromy matrix
  cout << "D_x phi(t,u) = " << (IMatrix)set << endl;

  ITimeMap::SolutionCurve solution(0.);
  C1HORect2Set s(u);
  // integrate and record trajectory
  tm(2.,s,solution);
  cout << "solution(1) = " << solution(1) << endl;
  cout << "D_x solution(1.5) = " << solution.derivative(1.5) << endl;
  return 0;
}
```

```
/* Output of the program:
phi(2,u)={[3.57516, 3.57516],[-0.797643, -0.797643],[27.9102, 27.9102]
D_x phi(t,u) = {
{[-1.95841, -1.95841],[-3.28406, -3.28406],[-0.614346, -0.614346]},
{[1.17314, 1.17314],[1.71972, 1.71972],[0.813136, 0.813136]},
{[-4.75217, -4.75217],[-7.68436, -7.68436],[-2.00251, -2.00251]}
}
solution(1) = {[11.8084, 11.8084],[16.4144, 16.4144],[25.3, 25.3]}
D_x solution(1.5) = {
{[0.974059, 0.974059],[1.87338, 1.87338],[-0.125996, -0.125996]},
{[-0.276187, -0.276187],[-0.249576, -0.249576],[-0.470697, -0.470697]}
{[3.59364, 3.59364],[6.48145, 6.48145],[0.30862, 0.30862]}
}*/
```

```
/** Example of computation of derivative of Poincare map **/
#include <iostream>
#include "capd/capdlib.h"
using namespace capd;
using namespace std;
int main(){
  cout.precision(5);
  IMap lorenz("var:x,y,z;fun:10*(y-x),x*(28-z)-y,x*y-8*z/3;");
  IOdeSolver solver(lorenz,20);         // ODE integrator
  ICoordinateSection section(3,2,27.); // section is z=27
  IPoincareMap pm(solver,section);

  C1HORect2Set set(IVector({1,5,27}));
  IMatrix Dphi(3,3);
  IVector P = pm(set,Dphi);
  // recompute derivative of flow to derivative of Poincare map
  IMatrix DP = pm.computeDP(P,Dphi);
  cout << "P(1,5,27)=" << P << endl;
  cout << "DP(1,5,27)=" << DP << endl;
}
/* Output:
P(1,5,27)={[11.361, 11.361],[14.338, 14.338],[27, 27]}
DP(1,5,27)={
{[-0.19415, -0.19415],[-0.46759, -0.46759],[-0.087997, -0.087997]},
{[-0.4121, -0.4121],[-0.96593, -0.96593],[-0.18836, -0.18836]},
{[-1.37e-13, 1.3706e-13],[-5.2713e-13, 5.2758e-13],[-2.5335e-13, 2.533
}
*/
```

Case study

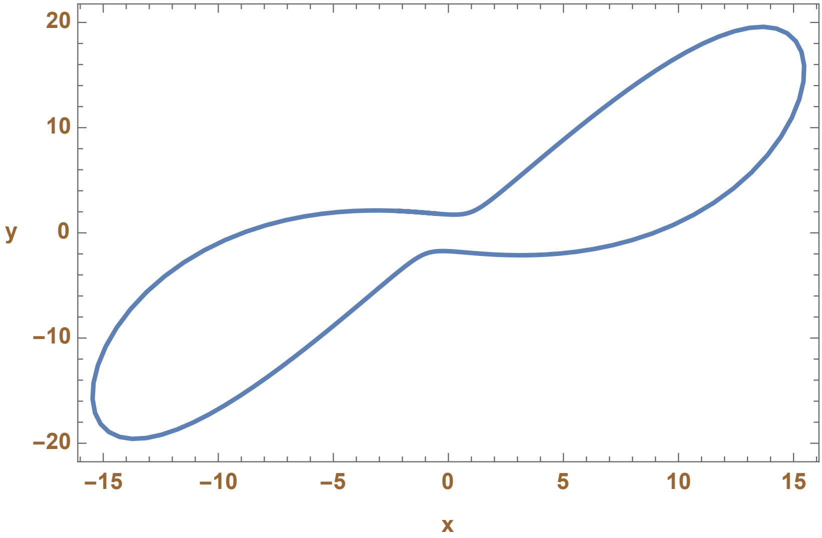
- hyperbolic periodic orbits
- solving BVP
- hyperbolic chaos in the Rössler system

1. I. Walawska, DW, *An implicit algorithm for validated enclosures of the solutions to variational equations for ODEs*, Appl. Math. Comput., 291C (2016), 303-322
2. T. Kapela, M. Mrozek, D. Wilczak, P. Zgliczyński, *CAPD::DynSys: a flexible C++ toolbox for rigorous numerical analysis of dynamical systems*, Communications in Nonlinear Science and Numerical Simulation (2021).

Example (Toy example: Lorenz system)

$$x' = 10(y - x), \qquad y' = x(28 - z) - y, \qquad z' = xy - 8z/3$$

Goal: prove that there is a hyperbolic periodic orbit



Methodology:

- Poincaré map

$$\Pi = \{(x, y, 27) : x, y \in \mathbb{R}\}, \qquad P : \Pi \to \Pi$$

- Prove (interval Newton operator) that the function

$$f(u) := P^4(u) - u$$

has zero

Data:

$$u_0 = (-2.14737, 2.07805)$$

$$[r] = ([-10^{-5}, 10^{-5}], [-10^{-5}, 10^{-5}])$$

Check

$$[N] := -\left(P^4(u_0 + [r]) - \mathrm{Id}\right)^{-1} \cdot \left(P^4(u_0) - u_0\right) \subset [r]$$

Compute eigenvalues of $DP^4(u_0 + [r])$

(Could be improved to $P^4(u_0 + [N])$)

Methodology:

- Poincaré map

$$\Pi = \{(x, y, 27) : x, y \in \mathbb{R}\}, \qquad P : \Pi \to \Pi$$

- Prove (interval Newton operator) that the function

$$f(u) := P^4(u) - u$$

has zero

Data:

$$u_0 = (-2.14737, 2.07805)$$

$$[r] = ([-10^{-5}, 10^{-5}], [-10^{-5}, 10^{-5}])$$

Check

$$[N] := -\left(P^4(u_0 + [r]) - \mathrm{Id}\right)^{-1} \cdot \left(P^4(u_0) - u_0\right) \subset [r]$$

Compute eigenvalues of $DP^4(u_0 + [r])$

(Could be improved to $P^4(u_0 + [N])$)

Methodology:

- Poincaré map

$$\Pi = \{(x, y, 27) : x, y \in \mathbb{R}\}, \qquad P : \Pi \to \Pi$$

- Prove (interval Newton operator) that the function

$$f(u) := P^4(u) - u$$

has zero

Data:

$$u_0 = (-2.14737, 2.07805)$$
$$[r] = ([-10^{-5}, 10^{-5}], [-10^{-5}, 10^{-5}])$$

Check

$$[N] := -\left(P^4(u_0 + [r]) - \mathrm{Id}\right)^{-1} \cdot \left(P^4(u_0) - u_0\right) \subset [r]$$

Compute eigenvalues of $DP^4(u_0 + [r])$

(Could be improved to $P^4(u_0 + [N])$)

Methodology:

- Poincaré map

$$\Pi = \{(x, y, 27) : x, y \in \mathbb{R}\}, \qquad P : \Pi \to \Pi$$

- Prove (interval Newton operator) that the function

$$f(u) := P^4(u) - u$$

has zero

Data:

$$u_0 = (-2.14737, 2.07805)$$

$$[r] = ([-10^{-5}, 10^{-5}], [-10^{-5}, 10^{-5}])$$

Check

$$[N] := -\left(P^4(u_0 + [r]) - \mathrm{Id}\right)^{-1} \cdot \left(P^4(u_0) - u_0\right) \subset [r]$$

Compute eigenvalues of $DP^4(u_0 + [r])$

(Could be improved to $P^4(u_0 + [N])$)

Methodology:

- Poincaré map

$$\Pi = \{(x, y, 27) : x, y \in \mathbb{R}\}, \qquad P : \Pi \to \Pi$$

- Prove (interval Newton operator) that the function

$$f(u) := P^4(u) - u$$

  has zero

Data:

$$u_0 = (-2.14737, 2.07805)$$
$$[r] = ([-10^{-5}, 10^{-5}], [-10^{-5}, 10^{-5}])$$

Check

$$[N] := -\left(P^4(u_0 + [r]) - \mathrm{Id}\right)^{-1} \cdot \left(P^4(u_0) - u_0\right) \subset [r]$$

Compute eigenvalues of $DP^4(u_0 + [r])$

(Could be improved to $P^4(u_0 + [N])$)

```
int main(){
  IMap lorenz("var:x,y,z;fun:10*(y-x),x*(28-z)-y,x*y-8*z/3;");
  IOdeSolver solver(lorenz,20);          // ODE integrator
  ICoordinateSection section(3,2,27.); // section is z=27
  IPoincareMap pm(solver,section);
  // very rough approximation of a periodic point
  IVector u0({-2.14737, 2.07805, 27});
  IVector r = IVector({1.,1.,0.})*interval(-1e-5,1e-5);
  C0HOTripletonSet s0(u0);
  // compute fu0:=P^4(u0)-u0 and project it onto (x,y)
  IVector fu0( 2, (pm(s0,4) - u0).begin() );
  // compute derivative
  C1HORect2Set s1(u0+r);
  IMatrix Dphi(3,3);
  IVector Pu = pm(s1,Dphi,4);
  IMatrix DP = pm.computeDP(Pu,Dphi);
  // projection of DP^4(u0+r)-Id onto 2D subspace
  IMatrix M({{DP(1,1)-1,DP(1,2)},{DP(2,1),DP(2,2)-1}});
  // enclose -(DP^4(u0+r)-Id)^{-1}*(P^4(u0)-u0)
  IVector N = - matrixAlgorithms::gauss(M,fu0);
  cout << "validated? " << subset(N,IVector(2,r.begin())) << endl;
  cout << "DP=" << IMatrix({{DP(1,1),DP(1,2)},{DP(2,1),DP(2,2)}});
  // explicit formula for eigenvalues of a 2x2 matrix
  interval t = sqrt(4*DP(2,1)*DP(1,2) + sqr(DP(1,1)-DP(2,2)));
  cout << "\nlambda1=" << 0.5*(DP(1,1)+DP(2,2)-t);
  cout << "\nlambda2=" << 0.5*(DP(1,1)+DP(2,2)+t);
}
```

```
/* Output of the program:
validated? 1
DP={
{[1.01919, 1.0261],[2.72595, 2.74535]},
{[1.37796, 1.38107],[3.68603, 3.69458]}
}
lambda1=[-0.00979719, 0.00979936]
lambda2=[4.70315, 4.72275]
*/
```

Example (Rössler system)

$$x' = -(y+z), \qquad y' = x + 0.2y, \qquad z' = 0.2 + z(x - 5.7)$$

Goal: localize with high accuracy three periodic orbits

Methodology:

- Poincaré map

$$\Pi = \{(0, y, z) : x, y \in \mathbb{R}\}, \qquad P : \Pi \to \Pi$$

- Prove (interval Newton operator) that the function

$$f(u) := P^n(u) - u$$

  has zero

```
/* Output of the program:
(validated?, accuracy) = true, 7.87471e-41
(validated?, accuracy) = true, 3.35732e-41
(validated?, accuracy) = true, 3.36216e-41 */
```

Remark 1: one can check that these orbits are of period 1, 2, 3.

Remark 2: accuracy – diameter of returned enclosure

Example (Rössler system)

$$x' = -(y+z), \qquad y' = x + 0.2y, \qquad z' = 0.2 + z(x - 5.7)$$

Goal: localize with high accuracy three periodic orbits

Methodology:

- Poincaré map

$$\Pi = \{(0,y,z) : x, y \in \mathbb{R}\}, \qquad P : \Pi \to \Pi$$

- Prove (interval Newton operator) that the function

$$f(u) := P^n(u) - u$$

  has zero

```
/* Output of the program:
(validated?, accuracy) = true, 7.87471e-41
(validated?, accuracy) = true, 3.35732e-41
(validated?, accuracy) = true, 3.36216e-41 */
```

Remark 1: one can check that these orbits are of period 1, 2, 3.

Remark 2: accuracy – diameter of returned enclosure

Example (Rössler system)

$$x' = -(y+z), \qquad y' = x + 0.2y, \qquad z' = 0.2 + z(x - 5.7)$$

Goal: localize with high accuracy three periodic orbits

Methodology:

- Poincaré map

$$\Pi = \{(0, y, z) : x, y \in \mathbb{R}\}, \qquad P : \Pi \to \Pi$$

- Prove (interval Newton operator) that the function

$$f(u) := P^n(u) - u$$

  has zero

```
/* Output of the program:
(validated?, accuracy) = true, 7.87471e-41
(validated?, accuracy) = true, 3.35732e-41
(validated?, accuracy) = true, 3.36216e-41 */
```

Remark 1: one can check that these orbits are of period 1, 2, 3.

Remark 2: accuracy – diameter of returned enclosure

Example (Rössler system)

$$x' = -(y+z), \qquad y' = x + 0.2y, \qquad z' = 0.2 + z(x - 5.7)$$

Goal: localize with high accuracy three periodic orbits

Methodology:

- Poincaré map

$$\Pi = \{(0, y, z) : x, y \in \mathbb{R}\}, \qquad P : \Pi \to \Pi$$

- Prove (interval Newton operator) that the function

$$f(u) := P^n(u) - u$$

  has zero

```
/* Output of the program:
(validated?, accuracy) = true, 7.87471e-41
(validated?, accuracy) = true, 3.35732e-41
(validated?, accuracy) = true, 3.36216e-41 */
```

Remark 1: one can check that these orbits are of period 1, 2, 3.

Remark 2: accuracy – diameter of returned enclosure

Example (Rössler system)

$$x' = -(y+z), \qquad y' = x + 0.2y, \qquad z' = 0.2 + z(x - 5.7)$$

Goal: localize with high accuracy three periodic orbits

Methodology:

- Poincaré map

$$\Pi = \{(0, y, z) : x, y \in \mathbb{R}\}, \qquad P : \Pi \to \Pi$$

- Prove (interval Newton operator) that the function

$$f(u) := P^n(u) - u$$

  has zero

```
/* Output of the program:
(validated?, accuracy) = true, 7.87471e-41
(validated?, accuracy) = true, 3.35732e-41
(validated?, accuracy) = true, 3.36216e-41 */
```

Remark 1: one can check that these orbits are of period 1, 2, 3.

Remark 2: accuracy – diameter of returned enclosure

Example (Rössler system)

$$x' = -(y+z), \qquad y' = x + 0.2y, \qquad z' = 0.2 + z(x - 5.7)$$

Goal: localize with high accuracy three periodic orbits

Methodology:

- Poincaré map

$$\Pi = \{(0, y, z) : x, y \in \mathbb{R}\}, \qquad P : \Pi \to \Pi$$

- Prove (interval Newton operator) that the function

$$f(u) := P^n(u) - u$$

  has zero

```
/* Output of the program:
(validated?, accuracy) = true, 7.87471e-41
(validated?, accuracy) = true, 3.35732e-41
(validated?, accuracy) = true, 3.36216e-41 */
```

Remark 1: one can check that these orbits are of period 1, 2, 3.

Remark 2: accuracy – diameter of returned enclosure

```
void check(MpFloat y, MpFloat z, int n, double e=1e-22){
 MpIMap rossler("var:x,y,z;fun:-(y+z),x+0.2*y,0.2+z*(x-5.7);");
 MpIOdeSolver solver(rossler,80);        // ODE integrator
 MpICoordinateSection section(3,0.);    // section is x=0
 MpIPoincareMap pm(solver,section, poincare::MinusPlus);
 // approximate periodic point and a ball
 MpIVector u0({MpInterval(0.),y,z});
 MpIVector r({MpInterval(0.),MpInterval(-e,e),MpInterval(-e,e)});
 MpC0TripletonSet s0(u0);
 // compute P^n(u0)-u0 and project it onto (y,z)
 MpIVector fu0( 2, (pm(s0,n) - u0).begin() + 1 );
 MpC1Rect2Set s1(u0+r);    // compute derivative on u0+r
 MpIMatrix Dphi(3,3);
 MpIVector u = pm(s1,Dphi,n);
 MpIMatrix DP = pm.computeDP(u,Dphi);
 // projection of DP^n(u0+r)-Id onto 2D subspace
 MpIMatrix M({{DP(2,2)-1.,DP(2,3)},{DP(3,2),DP(3,3)-1.}});
 // enclose -(DP^n(u0+r)-Id)^{-1}*(P^n(u0)-u0)
 MpIVector N = - matrixAlgorithms::gauss(M,fu0);
 cout << boolalpha << "\n(validated?, accuracy) = "
      << subset(N,MpIVector(2,r.begin()+1)) << ", " << maxWidth(N);
}
int main(){
 MpFloat::setDefaultPrecision(200);
 check("-8.38094174282987645183593","0.02959006063066710383300745",1);
 check("-5.42407382266520422640036","0.031081210807644620332608",2);
 check("-6.23315862853797465596029","0.03064011165816057058379228",3);
}
```

Boundary value problems for ODEs of any type

Second order equation:

$$x'' = f(x, x')$$

BVPs can be transformed to $F(u) = 0$

Dirichlet BVP:

$$x(0) = A,\ x(T) = B \qquad \rightsquigarrow \qquad F(x') = \pi_x\left(\varphi(T, (A, x'))\right) - B$$

Neumann BVP:

$$x'(0) = A,\ x'(T) = B \qquad \rightsquigarrow \qquad F(x) = \pi_{x'}\left(\varphi(T, (x, A))\right) - B$$

Solve by interval Newton (Krawczyk) method

Boundary value problems for ODEs of any type

Second order equation:

$$x'' = f(x, x')$$

BVPs can be transformed to $F(u) = 0$

Dirichlet BVP:

$$x(0) = A,\ x(T) = B \qquad \rightsquigarrow \qquad F(x') = \pi_x\left(\varphi(T, (A, x'))\right) - B$$

Neumann BVP:

$$x'(0) = A,\ x'(T) = B \qquad \rightsquigarrow \qquad F(x) = \pi_{x'}\left(\varphi(T, (x, A))\right) - B$$

Solve by interval Newton (Krawczyk) method

Boundary value problems for ODEs of any type

Second order equation:

$$x'' = f(x, x')$$

BVPs can be transformed to $F(u) = 0$

Dirichlet BVP:

$$x(0) = A,\ x(T) = B \qquad \rightsquigarrow \qquad F(x') = \pi_x\left(\varphi(T, (A, x'))\right) - B$$

Neumann BVP:

$$x'(0) = A,\ x'(T) = B \qquad \rightsquigarrow \qquad F(x) = \pi_{x'}\left(\varphi(T, (x, A))\right) - B$$

Solve by interval Newton (Krawczyk) method

Boundary value problems for ODEs of any type

Second order equation:

$$x'' = f(x, x')$$

BVPs can be transformed to $F(u) = 0$

Dirichlet BVP:

$$x(0) = A,\ x(T) = B \qquad \rightsquigarrow \qquad F(x') = \pi_x\left(\varphi(T, (A, x'))\right) - B$$

Neumann BVP:

$$x'(0) = A,\ x'(T) = B \qquad \rightsquigarrow \qquad F(x) = \pi_{x'}\left(\varphi(T, (x, A))\right) - B$$

Solve by interval Newton (Krawczyk) method

Boundary value problems for ODEs of any type

Second order equation:

$$x'' = f(x, x')$$

BVPs can be transformed to $F(u) = 0$

Dirichlet BVP:

$$x(0) = A,\; x(T) = B \qquad \rightsquigarrow \qquad F(x') = \pi_x\left(\varphi(T, (A, x'))\right) - B$$

Neumann BVP:

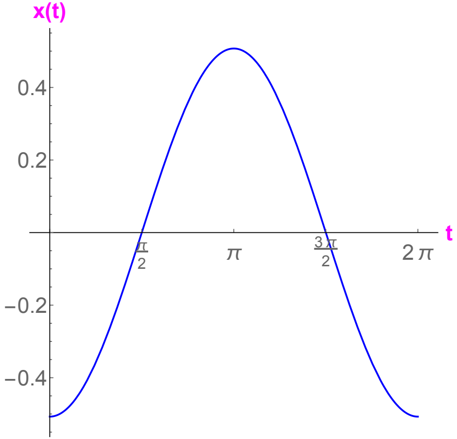
$$x'(0) = A,\; x'(T) = B \qquad \rightsquigarrow \qquad F(x) = \pi_{x'}\left(\varphi(T, (x, A))\right) - B$$

Solve by interval Newton (Krawczyk) method

Example (Taken from Nakao, J. Math. Anal. App. 1992)

$$x'' = -0.1x - 0.1x^3 - 0.4464\cos t$$

Find solution to $x'(0) = x'(2\pi) = 0$.



Method: solve using interval Newton method

$$F(r_1) = \varphi_{\dot{x}}(2\pi, (-0.5072 + r_1, 0)) = 0$$

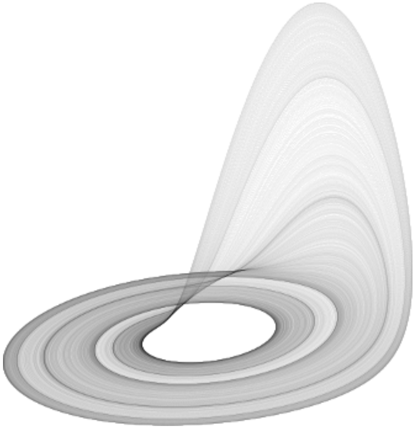
```
/** Example of solving BVP: x'(0)=x'(2pi)=0 **/
#include <iostream>
#include "capd/capdlib.h"
using namespace capd;
using namespace std;
int main(){
  IMap f("par:a;time:t;var:x,dx;fun:dx,-x*(1+x^2)/10 + a*cos(t);");
  f.setParameter("a",interval(4464)/interval(10000));
  IOdeSolver solver(f,20);  // ODE integrator
  ITimeMap tm(solver);      // class for long time integration

  IVector u0({-0.5072,0.});
  IVector r({interval(-1e-5,1e-5),0.});
  C0HOTripletonSet s0(u0);
  C1HORect2Set s(u0+r);
  // integrate until T=2*pi
  IVector y = tm(2.*interval::pi(),s0);
  tm(2.*interval::pi(),s);
  // solve equation F(r1) := proj_{x'}(phi(2pi,u0+(r1,0))) = 0
  interval N = - y[1]/((IMatrix)s)(2,1);
  cout << "(N,r1)=(" << N << "," << r[0] << ")"<< endl;
  cout << "subset(N,r)? = " << boolalpha << subset(N,r[0]) << endl;
  return 0;
}
/* Output:
(N,r1)=([-3.84493e-05, -2.09823e-05],[-0.1, 0.1])
subset(N,r)? = true
*/
```

Example (Rössler system)

$$x' = -(y+z), \qquad y' = x + 0.2y, \qquad z' = 0.2 + z(x - 5.7)$$

Goal 1: there is a compact, connected and isolated invariant $\mathcal{A}$ - an attractor.

Goal 2: There is $\mathcal{H} \subset \mathcal{A}$ - invariant, uniformly hyperbolic and chaotic subset.

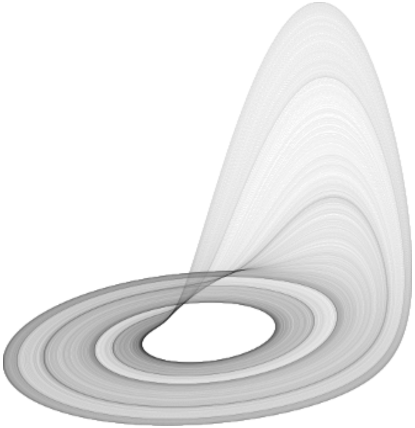


Example (Rössler system)

$$x' = -(y+z), \qquad y' = x + 0.2y, \qquad z' = 0.2 + z(x - 5.7)$$

Goal 1: there is a compact, connected and isolated invariant $\mathcal{A}$ - an attractor.

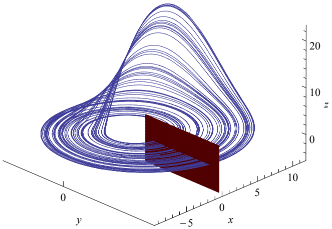
Goal 2: There is $\mathcal{H} \subset \mathcal{A}$ - invariant, uniformly hyperbolic and chaotic subset.



Example: attractor in the Rössler system

Settings:

$$\Pi = \{(0, y, z) : y, z \in \mathbb{R}, x' > 0\} \quad - \quad \text{Poincaré section}$$
$$P : \Pi \to \Pi \quad - \quad \text{Poincaré map}$$



Methodology: Show that there is a rectangle

$$W = [y_1, y_2] \times [z_1, z_2]$$

such that

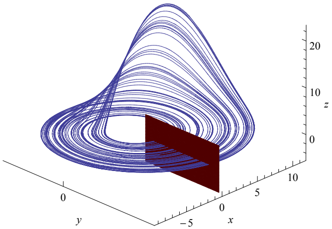
$$P(W) \subset W.$$

Then $\mathcal{A} := \bigcap_{n>0} P^n(W)$ is a compact, connected invariant set.

Example: attractor in the Rössler system

Settings:

$\Pi = \{(0, y, z) : y, z \in \mathbb{R}, x' > 0\}$	–	Poincaré section
$P : \Pi \to \Pi$	–	Poincaré map



Methodology: Show that there is a rectangle

$$W = [y_1, y_2] \times [z_1, z_2]$$

such that

$$P(W) \subset W.$$

Then $\mathcal{A} := \bigcap_{n>0} P^n(W)$ is a compact, connected invariant set.

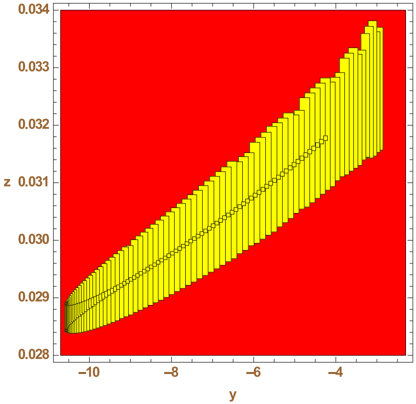
Example: attractor in the Rössler system

Data (from simulation):

$$W = [-10.7, -2.3] \times [0.028, 0.034]$$

Computations:

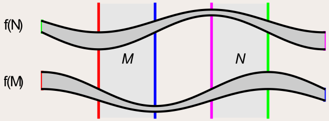
- subdivide $W = \bigcup_{i=1}^{200} W_i$
- check that $P(W_i) \subset W$ for $i = 1, \ldots, 200$



```
#include <iostream>
#include "capd/capdlib.h"
using namespace capd;

int main(){
  IMap vf("var:x,y,z;fun:-(y+z),x+0.2*y,0.2+z*(x-5.7);");
  IOdeSolver solver(vf, 20);
  ICoordinateSection section(3, 0); // section x=0, x'>0
  IPoincareMap pm(solver, section, poincare::MinusPlus);

  // Coordinates of the trapping region
  const double B = 0.028, T = 0.034, L = -10.7, R = -2.3;

  // Subdivide uniformly onto 200 rectangles
  const int N = 160;
  bool result = true;
  interval p = (interval(R) - interval(L)) / N;
  for (int i = 0; i < N and result; ++i) {
    IVector x ({0., L + interval(i,i+1)*p, interval(B, T)});
    C0HOTripletonSet s(x);
    IVector u = pm(s);
    result = result and u[2]>B and u[2]<T and u[1]>L and u[1]<R;
    if(!result)
      std::cout << "Inclusion not satisfied:\n" << u << std::endl;
  }
  std::cout << "Existence of attractor: " << result << std::endl;
  return 0;
}
```

Theorem

- *Let $Q = DiagonalMatrix\{\lambda, \mu\}$ with $\mu < 0 < \lambda$*
- *N, M disjoint rectangles aligned to the axes*
- *$f : N \cup M \to \mathbb{R}^2$ is smooth*



- *$N \xrightarrow{f} N \xrightarrow{f} M \xrightarrow{f} M \xrightarrow{f} N.$*
- *$Df(x)^T Q Df(x) - Q$ is positive definite for $x \in N \cup M$*

Then

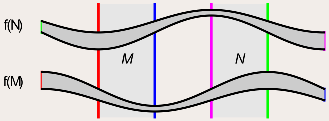
- *$\mathcal{H} = \mathrm{Inv}(f, N \cup M)$ is uniformly hyperbolic and the dynamics of f on $\mathcal{H}$ is conjugated to the full shift on two symbols*
- *every binifinite sequence of symbols of the form*

$$(\ldots, A, A, \{N, M\}^k, B, B, \ldots), \qquad A, B \in \{N, M\}$$

is realized by a connecting orbit joining unique fixed point $p_A \in A$ and $p_B \in B$

Theorem

- *Let $Q = DiagonalMatrix\{\lambda, \mu\}$ with $\mu < 0 < \lambda$*
- *N, M disjoint rectangles aligned to the axes*
- *$f : N \cup M \to \mathbb{R}^2$ is smooth*
- *$N \xrightarrow{f} N \xrightarrow{f} M \xrightarrow{f} M \xrightarrow{f} N.$*
- *$Df(x)^T QDf(x) - Q$ is positive definite for $x \in N \cup M$*



Then

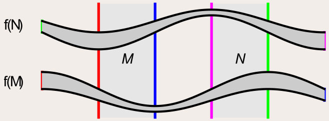
- *$\mathcal{H} = \mathrm{Inv}(f, N \cup M)$ is uniformly hyperbolic and the dynamics of f on $\mathcal{H}$ is conjugated to the full shift on two symbols*
- *every binifinite sequence of symbols of the form*

$$(\ldots, A, A, \{N, M\}^k, B, B, \ldots), \qquad A, B \in \{N, M\}$$

is realized by a connecting orbit joining unique fixed point $p_A \in A$ and $p_B \in B$

Theorem

- *Let $Q = \text{DiagonalMatrix}\{\lambda, \mu\}$ with $\mu < 0 < \lambda$*
- *N, M disjoint rectangles aligned to the axes*
- *$f : N \cup M \to \mathbb{R}^2$ is smooth*



- *$N \xrightarrow{f} N \xrightarrow{f} M \xrightarrow{f} M \xrightarrow{f} N$.*
- *$Df(x)^T QDf(x) - Q$ is positive definite for $x \in N \cup M$*

Then

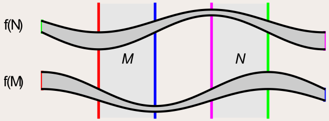
- *$\mathcal{H} = \mathrm{Inv}(f, N \cup M)$ is uniformly hyperbolic and the dynamics of f on $\mathcal{H}$ is conjugated to the full shift on two symbols*
- *every binifinite sequence of symbols of the form*

$$(\ldots, A, A, \{N, M\}^k, B, B, \ldots), \qquad A, B \in \{N, M\}$$

is realized by a connecting orbit joining unique fixed point $p_A \in A$ and $p_B \in B$

Theorem

- *Let* $Q = \textit{DiagonalMatrix}\{\lambda, \mu\}$ *with* $\mu < 0 < \lambda$
- N, M *disjoint rectangles aligned to the axes*
- $f : N \cup M \to \mathbb{R}^2$ *is smooth*
- $N \overset{f}{\Longrightarrow} N \overset{f}{\Longrightarrow} M \overset{f}{\Longrightarrow} M \overset{f}{\Longrightarrow} N.$



- $Df(x)^T Q Df(x) - Q$ *is positive definite for* $x \in N \cup M$

Then

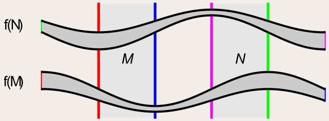
- $\mathcal{H} = \mathrm{Inv}(f, N \cup M)$ *is uniformly hyperbolic and the dynamics of* f *on* $\mathcal{H}$ *is conjugated to the full shift on two symbols*
- *every binifinite sequence of symbols of the form*

$$(\dots, A, A, \{N, M\}^k, B, B, \dots), \qquad A, B \in \{N, M\}$$

is realized by a connecting orbit joining unique fixed point $p_A \in A$ *and* $p_B \in B$

Theorem

- *Let* $Q = \mathit{DiagonalMatrix}\{\lambda, \mu\}$ *with* $\mu < 0 < \lambda$
- N, M *disjoint rectangles aligned to the axes*
- $f : N \cup M \to \mathbb{R}^2$ *is smooth*
- $N \xRightarrow{f} N \xRightarrow{f} M \xRightarrow{f} M \xRightarrow{f} N.$



- $Df(x)^T QDf(x) - Q$ *is positive definite for* $x \in N \cup M$

Then

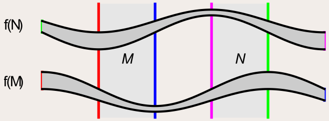
- $\mathcal{H} = \mathrm{Inv}(f, N \cup M)$ *is uniformly hyperbolic and the dynamics of* f *on* $\mathcal{H}$ *is conjugated to the full shift on two symbols*
- *every binifinite sequence of symbols of the form*

$$(\ldots, A, A, \{N, M\}^k, B, B, \ldots), \qquad A, B \in \{N, M\}$$

is realized by a connecting orbit joining unique fixed point $p_A \in A$ *and* $p_B \in B$

Theorem

- *Let* $Q = \mathit{DiagonalMatrix}\{\lambda, \mu\}$ *with* $\mu < 0 < \lambda$
- N, M *disjoint rectangles aligned to the axes*
- $f : N \cup M \to \mathbb{R}^2$ *is smooth*
- $N \overset{f}{\Longrightarrow} N \overset{f}{\Longrightarrow} M \overset{f}{\Longrightarrow} M \overset{f}{\Longrightarrow} N.$



- $Df(x)^T QDf(x) - Q$ *is positive definite for* $x \in N \cup M$

Then

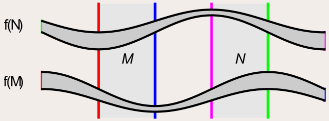
- $\mathcal{H} = \mathrm{Inv}(f, N \cup M)$ *is uniformly hyperbolic and the dynamics of* f *on* $\mathcal{H}$ *is conjugated to the full shift on two symbols*
- *every binifinite sequence of symbols of the form*

$$(\ldots, A, A, \{N, M\}^k, B, B, \ldots), \qquad A, B \in \{N, M\}$$

is realized by a connecting orbit joining unique fixed point $p_A \in A$ *and* $p_B \in B$

Theorem

- *Let* $Q = \mathit{DiagonalMatrix}\{\lambda, \mu\}$ *with* $\mu < 0 < \lambda$
- N, M *disjoint rectangles aligned to the axes*
- $f : N \cup M \to \mathbb{R}^2$ *is smooth*
- $N \overset{f}{\Longrightarrow} N \overset{f}{\Longrightarrow} M \overset{f}{\Longrightarrow} M \overset{f}{\Longrightarrow} N$.



- $Df(x)^T Q Df(x) - Q$ *is positive definite for* $x \in N \cup M$

Then

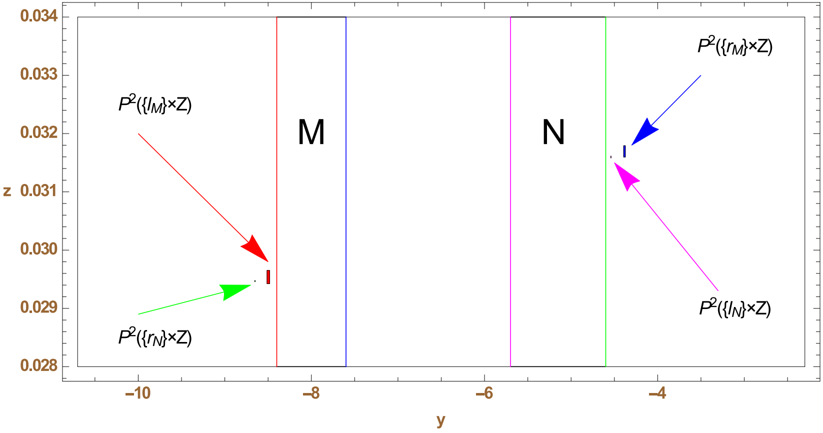
- $\mathcal{H} = \mathrm{Inv}(f, N \cup M)$ *is uniformly hyperbolic and the dynamics of* f *on* $\mathcal{H}$ *is conjugated to the full shift on two symbols*
- *every binifinite sequence of symbols of the form*

$$(\dots, A, A, \{N, M\}^k, B, B, \dots), \qquad A, B \in \{N, M\}$$

is realized by a connecting orbit joining unique fixed point $p_A \in A$ *and* $p_B \in B$

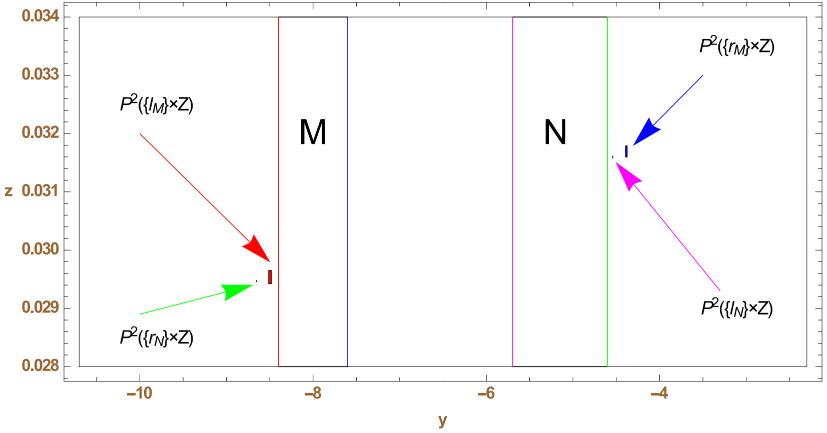
Data:

$$M = [l_M, r_M] \times Z = [-8.4, -7.6] \times [0.028, 0.034]$$
$$N = [l_N, r_N] \times Z = [-5.7, -4.6] \times [0.028, 0.034]$$
$$Q = \texttt{DiagonalMatrix}\{1, -100\}$$



Data:

$$M = [l_M, r_M] \times Z = [-8.4, -7.6] \times [0.028, 0.034]$$
$$N = [l_N, r_N] \times Z = [-5.7, -4.6] \times [0.028, 0.034]$$
$$Q = \texttt{DiagonalMatrix}\{1, -100\}$$



```
#include <iostream>
#include "capd/capdlib.h"
using namespace capd;
using namespace std;

int main(){
  IMap vf("var:x,y,z;fun:-(y+z),x+0.2*y,0.2+z*(x-5.7);");
  IOdeSolver solver(vf, 20);
  ICoordinateSection section(3, 0); // section x=0, x'>0
  IPoincareMap pm(solver, section, poincare::MinusPlus);

  // z-coordinate of the trapping region
  interval z(0.028,0.034);
  // Coordinates of M and N
  const double lM=-8.4, rM=-7.6, lN=-5.7, rN=-4.6;

  C0HOTripletonSet LM( IVector({0.,lM,z}) );
  C0HOTripletonSet RM( IVector({0.,rM,z}) );
  C0HOTripletonSet LN( IVector({0.,lN,z}) );
  C0HOTripletonSet RN( IVector({0.,rN,z}) );

  // Inequalities for the covering relations N=>N, N=>M, M=>M, M=>N.
  cout << "P^2(LM) < lM: " << ( pm(LM,2)[1] < lM ) << endl;
  cout << "P^2(RM) > rN: " << ( pm(RM,2)[1] > rN ) << endl;
  cout << "P^2(LN) > rN: " << ( pm(LN,2)[1] > rN ) << endl;
  cout << "P^2(RN) < lM: " << ( pm(RN,2)[1] < lM ) << endl;
  return 0;
}
```

```
#include <iostream>
#include "capd/capdlib.h"
using namespace capd;
using namespace std;
bool checkCC(IPoincareMap& pm, double y1, double y2, int N) {
  bool res = true;
  interval p = (interval(y2) - interval(y1)) / N;
  IMatrix Dphi(3,3);
  IMatrix Q({{0.,0.,0.},{0.,1,0.},{0.,0.,-100}});
  interval returnTime;
  for (int i = 0; i < N and res; ++i) {
    C1Rect2Set s({0.,y1+interval(i,i+1)*p,interval(0.028,0.034)});
    IVector y = pm(s, Dphi, returnTime, 2);
    IMatrix DP = pm.computeDP(y,Dphi);
    DP = Transpose(DP)*Q*DP - Q;
    res = res and DP(2,2)>0 and (DP(2,2)*DP(3,3)-sqr(DP(2,3)))>0;
  }
  return res;
}
int main(){
  IMap vf("var:x,y,z;fun:-(y+z),x+0.2*y,0.2+z*(x-5.7);");
  IOdeSolver solver(vf, 20);
  ICoordinateSection section(3, 0); // section x=0, x'>0
  IPoincareMap pm(solver, section, poincare::MinusPlus);
  const double lM=-8.4, rM=-7.6, lN=-5.7, rN=-4.6;
  cout << "Cone condition on M: " << checkCC(pm,lM,rM,80) << endl;
  cout << "Cone condition on N: " << checkCC(pm,lN,rN,40) << endl;
}
```

Hyperbolic chaotic attractor

- hyperbolic chaotic attractor in the Kuznetsov system

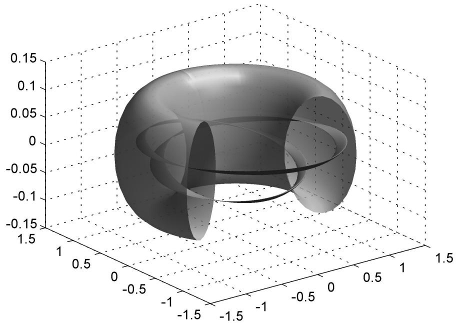
1. DW, *Uniformly hyperbolic attractor of the Smale-Williams type for a Poincaré map in the Kuznetsov system*, SIAM Journal on Applied Dynamical Systems 2010, Vol. 9, 1263–1283.

Example (Kuznetsov system)

$$\begin{cases} \dot{x} &= \omega_0 u, \\ \dot{u} &= -\omega_0 x + \left(A\cos(2\pi t/T) - x^2\right) u + (\varepsilon/\omega_0) y \cos(\omega_0 t), \\ \dot{y} &= 2\omega_0 v, \\ \dot{v} &= -2\omega_0 y + \left(-A\cos(2\pi t/T) - y^2\right) v + (\varepsilon/2\omega_0) x^2. \end{cases}$$

$\omega_0 = 2\pi$, $A = 5$, $T = 6$, $\varepsilon = 0.5$

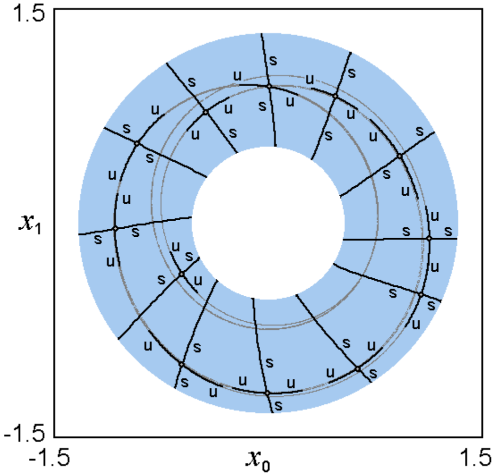
Click here to start animation



S.P. Kuznetsov, Example of a Physical System with a Hyperbolic Attractor of the Smale-Williams Type, Phys. Rev. Lett., 95, 2005, 144101.

Conjecture: the system has hyperbolic attractor for

$$\omega_0 = 2\pi, \quad A = 5, \quad T = 6, \quad \varepsilon = 0.5$$

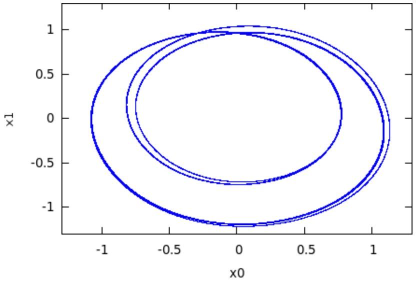


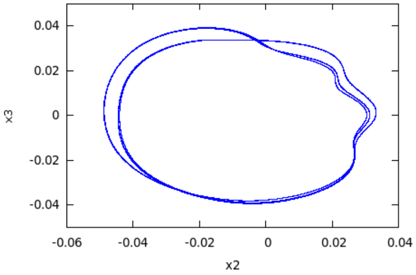
S.P. Kuznetsov and I.R. Sataev, Hyperbolic attractor in a system of coupled non-autonomous van der Pol oscillators: Numerical test for expanding and contracting cones, Phys. Lett. A 365, 97–104, (2007).

Theorem

Poincaré map: $P(x) = \varphi(T, x)$, $T = 6$ (period of vector field).
There is a compact, connected and explicitly given set $\mathcal{B}$ such that

1. *$P(\mathcal{B}) \subset \mathcal{B}$,*
2. *P is uniformly hyperbolic on $\mathcal{A} = \bigcap_{i>0} P^i(\mathcal{B})$ with one positive and three negative Lyapunov exponents.*
3. *$\mathcal{A}$ is a nontrivial continuum.*



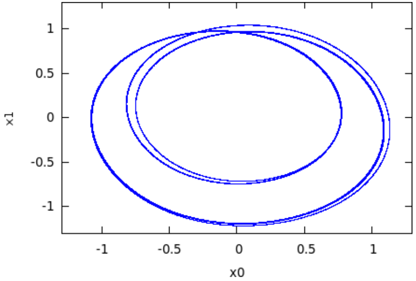


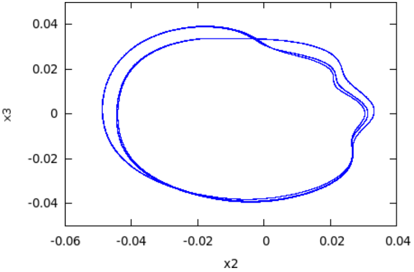
DW, Uniformly hyperbolic attractor of the Smale-Williams type for a Poincaré map in the Kuznetsov system, SIAM J. App. Dyn. Sys. 2010, Vol. 9, 1263–1283.

Theorem

Poincaré map: $P(x) = \varphi(T, x)$, $T = 6$ (period of vector field). There is a compact, connected and explicitly given set $\mathcal{B}$ such that

1. *$P(\mathcal{B}) \subset \mathcal{B}$,*
2. *P is uniformly hyperbolic on $\mathcal{A} = \bigcap_{i>0} P^i(\mathcal{B})$ with one positive and three negative Lyapunov exponents.*
3. *$\mathcal{A}$ is a nontrivial continuum.*



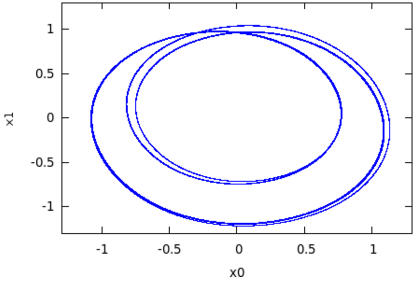


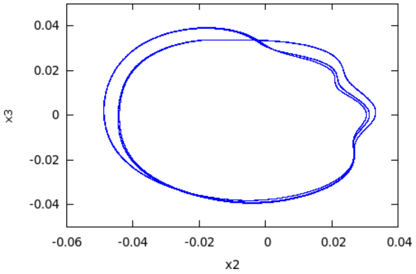
DW, Uniformly hyperbolic attractor of the Smale-Williams type for a Poincaré map in the Kuznetsov system, SIAM J. App. Dyn. Sys. 2010, Vol. 9, 1263–1283.

Theorem

Poincaré map: $P(x) = \varphi(T, x)$, $T = 6$ (period of vector field). There is a compact, connected and explicitly given set $\mathcal{B}$ such that

1. $P(\mathcal{B}) \subset \mathcal{B}$,
2. *P is uniformly hyperbolic on $\mathcal{A} = \bigcap_{i>0} P^i(\mathcal{B})$ with one positive and three negative Lyapunov exponents.*
3. *$\mathcal{A}$ is a nontrivial continuum.*



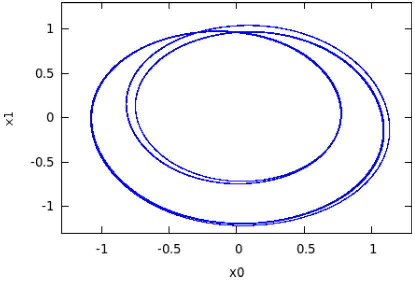


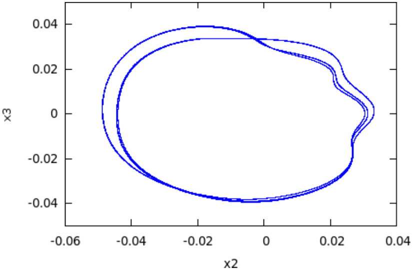
DW, Uniformly hyperbolic attractor of the Smale-Williams type for a Poincaré map in the Kuznetsov system, SIAM J. App. Dyn. Sys. 2010, Vol. 9, 1263–1283.

Theorem

Poincaré map: $P(x) = \varphi(T, x)$, $T = 6$ (period of vector field). There is a compact, connected and explicitly given set $\mathcal{B}$ such that

1. *$P(\mathcal{B}) \subset \mathcal{B}$,*
2. *P is uniformly hyperbolic on $\mathcal{A} = \bigcap_{i>0} P^i(\mathcal{B})$ with one positive and three negative Lyapunov exponents.*
3. *$\mathcal{A}$ is a nontrivial continuum.*



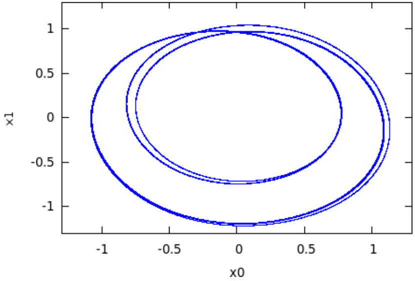


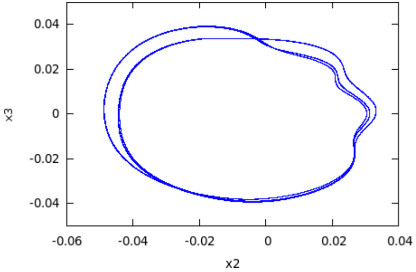
DW, Uniformly hyperbolic attractor of the Smale-Williams type for a Poincaré map in the Kuznetsov system, SIAM J. App. Dyn. Sys. 2010, Vol. 9, 1263–1283.

Theorem

Poincaré map: $P(x) = \varphi(T, x)$, $T = 6$ (period of vector field). There is a compact, connected and explicitly given set $\mathcal{B}$ such that

1. *$P(\mathcal{B}) \subset \mathcal{B}$,*
2. *P is uniformly hyperbolic on $\mathcal{A} = \bigcap_{i>0} P^i(\mathcal{B})$ with one positive and three negative Lyapunov exponents.*
3. *$\mathcal{A}$ is a nontrivial continuum.*





DW, Uniformly hyperbolic attractor of the Smale-Williams type for a Poincaré map in the Kuznetsov system, SIAM J. App. Dyn. Sys. 2010, Vol. 9, 1263–1283.

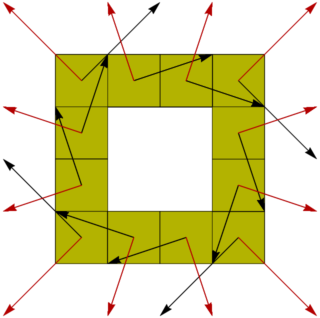
Theoretical background

General settings

- $M = \bigcup_{i=1}^{N} M_i$, where M_i are compact sets in $\mathbb{R}^n$.
- u, s - nonnegative, such that $n = u + s$
- C_i - a linear coordinate system assigned to the set M_i
- quadratic form on $\mathbb{R}^n$

$$Q(x, y) = \|x\|^2 - \|y\|^2, \quad x \in \mathbb{R}^u,\ y \in \mathbb{R}^s.$$

$\mathcal{M} = \left(Q, \{(M_i, C_i)\}_{i=1}^{N}\right)$ is called **cubical set with cones**.



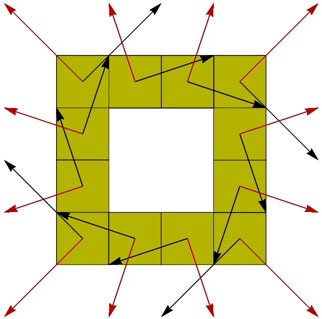
Theoretical background

General settings

- $M = \bigcup_{i=1}^{N} M_i$, where M_i are compact sets in $\mathbb{R}^n$.
- u, s - nonnegative, such that $n = u + s$
- C_i - a linear coordinate system assigned to the set M_i
- quadratic form on $\mathbb{R}^n$

$$Q(x, y) = \|x\|^2 - \|y\|^2, \quad x \in \mathbb{R}^u,\ y \in \mathbb{R}^s.$$

$\mathcal{M} = \left(Q, \{(M_i, C_i)\}_{i=1}^{N}\right)$ is called **cubical set with cones**.



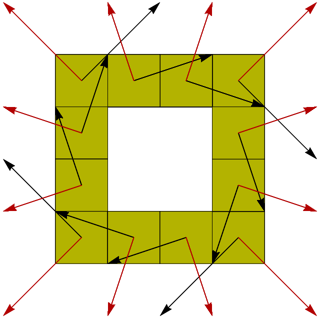
Theoretical background

General settings

- $M = \bigcup_{i=1}^{N} M_i$, where M_i are compact sets in $\mathbb{R}^n$.
- u, s - nonnegative, such that $n = u + s$
- C_i - a linear coordinate system assigned to the set M_i
- quadratic form on $\mathbb{R}^n$

$$Q(x, y) = \|x\|^2 - \|y\|^2, \quad x \in \mathbb{R}^u,\ y \in \mathbb{R}^s.$$

$\mathcal{M} = (Q, \{(M_i, C_i)\}_{i=1}^{N})$ is called **cubical set with cones**.



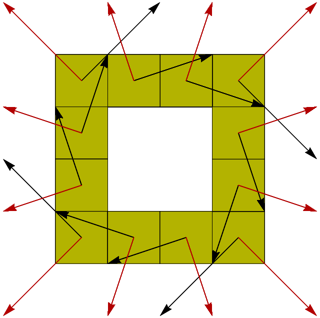
Theoretical background

General settings

- $M = \bigcup_{i=1}^{N} M_i$, where M_i are compact sets in $\mathbb{R}^n$.
- u, s - nonnegative, such that $n = u + s$
- C_i - a linear coordinate system assigned to the set M_i
- quadratic form on $\mathbb{R}^n$

$$Q(x, y) = \|x\|^2 - \|y\|^2, \quad x \in \mathbb{R}^u,\ y \in \mathbb{R}^s.$$

$\mathcal{M} = (Q, \{(M_i, C_i)\}_{i=1}^{N})$ is called **cubical set with cones**.



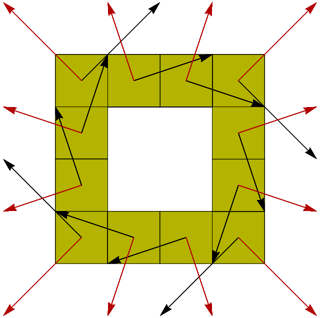
Theoretical background

General settings

- $M = \bigcup_{i=1}^{N} M_i$, where M_i are compact sets in $\mathbb{R}^n$.
- u, s - nonnegative, such that $n = u + s$
- C_i - a linear coordinate system assigned to the set M_i
- quadratic form on $\mathbb{R}^n$

$$Q(x,y) = \|x\|^2 - \|y\|^2, \quad x \in \mathbb{R}^u,\ y \in \mathbb{R}^s.$$

$\mathcal{M} = \left(Q, \{(M_i, C_i)\}_{i=1}^{N}\right)$ is called **cubical set with cones**.



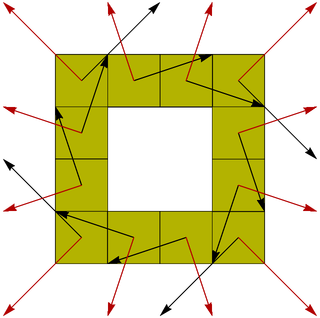
Theoretical background

General settings

- $M = \bigcup_{i=1}^{N} M_i$, where M_i are compact sets in $\mathbb{R}^n$.
- u, s - nonnegative, such that $n = u + s$
- C_i - a linear coordinate system assigned to the set M_i
- quadratic form on $\mathbb{R}^n$

$$Q(x, y) = \|x\|^2 - \|y\|^2, \quad x \in \mathbb{R}^u,\ y \in \mathbb{R}^s.$$

$\mathcal{M} = \left(Q, \{(M_i, C_i)\}_{i=1}^{N}\right)$ is called **cubical set with cones**.

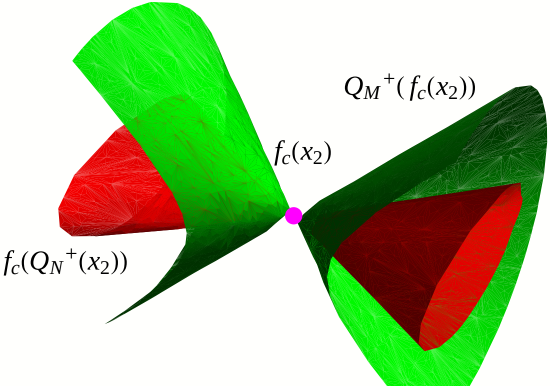


Definition

f is **strongly hyperbolic** on $\mathcal{M} = \left(Q, \{(M_i, C_i)\}_{i=1}^N\right)$ if for $z \in M_i$ and $j = 1, \ldots, N$ such that $f(M_i) \cap M_j \neq \emptyset$ the matrix

$$[C_j Df(z) C_i^{-1}]^T Q [C_j Df(z) C_i^{-1}] - Q$$

is positive definite.

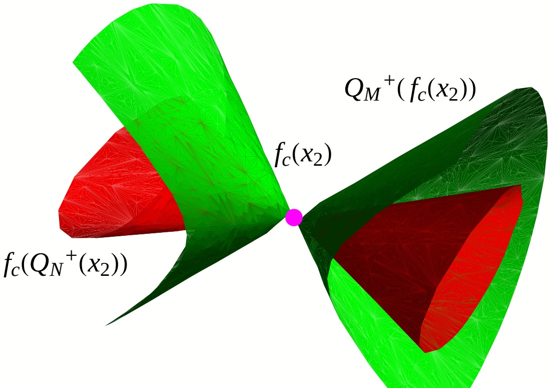


Definition

f is **strongly hyperbolic** on $\mathcal{M} = \left(Q, \{(M_i, C_i)\}_{i=1}^{N}\right)$ if for $z \in M_i$ and $j = 1, \ldots, N$ such that $f(M_i) \cap M_j \neq \emptyset$ the matrix

$$[C_j Df(z) C_i^{-1}]^T Q [C_j Df(z) C_i^{-1}] - Q$$

is positive definite.

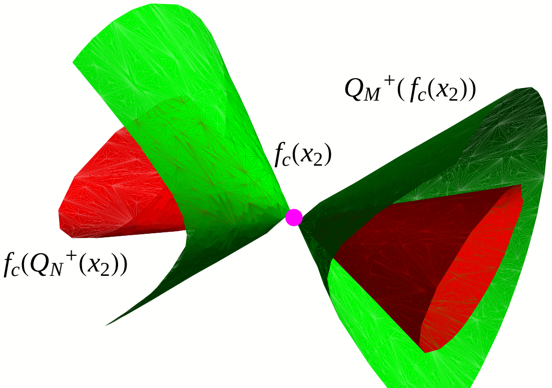


Definition

f is **strongly hyperbolic** on $\mathcal{M} = \left(Q, \{(M_i, C_i)\}_{i=1}^N\right)$ if for $z \in M_i$ and $j = 1, \ldots, N$ such that $f(M_i) \cap M_j \neq \emptyset$ the matrix

$$[C_j Df(z) C_i^{-1}]^T Q [C_j Df(z) C_i^{-1}] - Q$$

is positive definite.



Main tool for hyperbolicity

Theorem

Let

$$\mathcal{H} := \mathrm{Inv}(f, M) = \left\{x \in M : f^i(x) \in M, \text{for } i \in \mathbb{Z}\right\}.$$

Then

$$f \text{ is strongly hyperbolic on } \mathcal{M} = \left(Q, \{(M_i, C_i)\}_{i=1}^{N}\right)$$

$$\Downarrow$$

$$f \text{ is uniformly hyperbolic on } \mathcal{H}$$

Main tool for hyperbolicity

Theorem

Let

$$\mathcal{H} := \mathrm{Inv}(f, M) = \left\{x \in M : f^i(x) \in M, \textit{for } i \in \mathbb{Z}\right\}.$$

Then

$$f \textit{ is strongly hyperbolic on } \mathcal{M} = \left(Q, \{(M_i, C_i)\}_{i=1}^N\right)$$

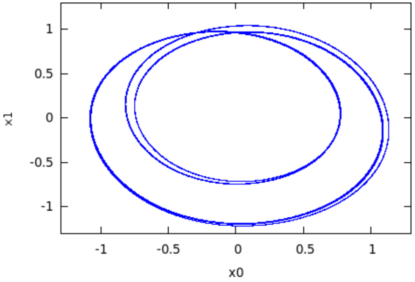
$$\Downarrow$$

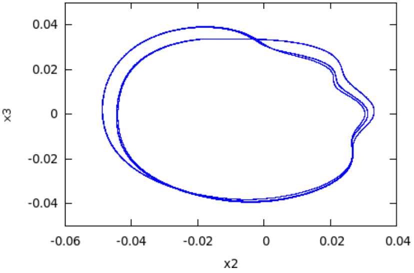
$$f \textit{ is uniformly hyperbolic on } \mathcal{H}$$

Time of computation:

Step	wall time (h:mm)	comments
enclosure of attractor	2:16 on 224 CPUs	7 970 392 boxes
strong hyperbolicity	4:24 on 224 CPUs	$\mathcal{C}^1$ computations

Quadratic form: $Q = \mathit{DiagMatrix}(1, -1, -1, -1)$





PDEs

- chaos and connecting orbits in infinite dimension

1. DW, P. Zgliczyński, *A geometric method for infinite-dimensional chaos: symbolic dynamics for the Kuramoto-Sivashinsky PDE on the line*, Journal of Differential Equations, Vol. 269 No. 10 (2020), 8509-8548.
2. DW, P. Zgliczyński, *A rigorous C^1 -algorithm for integration of dissipative PDEs based on automatic differentiation and the Taylor method*, in preparation.

Kuramoto-Sivashinsky equations

$$u_t = 2uu_x - u_{xx} - \nu u_{xxxx}$$

2π-periodic, odd

$$u(t,x) = -2\sum_{k=1}^{\infty} a_k(t)\sin(kx)$$

Infinite dimensional ODE

$$a'_k = k^2(1-\nu k^2)a_k - k\left(\sum_{n=1}^{k-1} a_n a_{k-n} - 2\sum_{n=1}^{\infty} a_n a_{n+k}\right)$$

Kuramoto-Sivashinsky equations

$$u_t = 2uu_x - u_{xx} - \nu u_{xxxx}$$

2π-periodic, odd

$$u(t,x) = -2\sum_{k=1}^{\infty} a_k(t)\sin(kx)$$

Infinite dimensional ODE

$$a_k' = k^2(1-\nu k^2)a_k - k\left(\sum_{n=1}^{k-1} a_n a_{k-n} - 2\sum_{n=1}^{\infty} a_n a_{n+k}\right)$$

Kuramoto-Sivashinsky equations

$$u_t = 2uu_x - u_{xx} - \nu u_{xxxx}$$

2π-periodic, odd

$$u(t,x) = -2\sum_{k=1}^{\infty} a_k(t)\sin(kx)$$

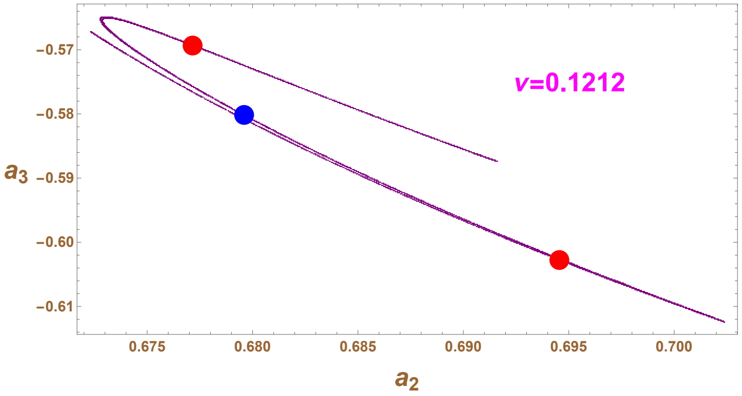
Infinite dimensional ODE

$$a_k' = k^2(1-\nu k^2)a_k - k\left(\sum_{n=1}^{k-1} a_n a_{k-n} - 2\sum_{n=1}^{\infty} a_n a_{n+k}\right)$$

Poincaré map

$$\Pi = \{a_1 = 0 \wedge a_1' < 0\}, \qquad P : \Pi \to \Pi$$

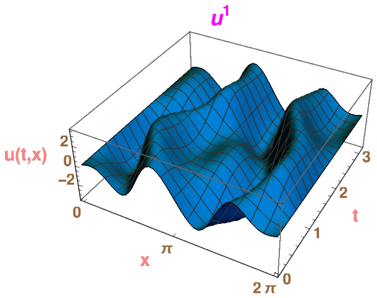
Observed chaotic attractor

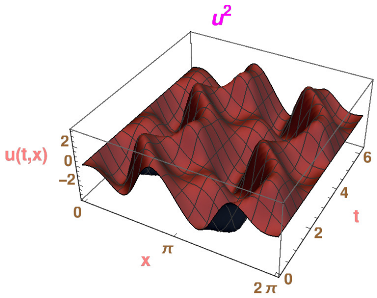


Theorem

Fix $\nu = 0.1212$*. There is an invariant set* $\mathcal{H}$ *such that*

- *the system on* $\mathcal{H}$ *is chaotic (symbolic dynamics)*
- $\mathcal{H}$ *possesses countable infinity of periodic orbits of arbitrary large prinicpal periods*
- *there are two hyperbolic periodic orbit* $\mathbf{u^1}, \mathbf{u^2} \subset \mathcal{H}$
- *there is countable infinity of connecting orbits between* $\mathbf{u^1}$ *and* $\mathbf{u^2}$ *in* $\mathcal{H}$ *in both directions*





Thank you for your attention

http://capd.ii.uj.edu.pl