

Stiffness Detection in Initial-Value ODEs

Raymond J. Spiteri

Department of Mathematics and Statistics

Faculty of Computer Science

Dalhousie University

Joint work with **Jesse Rusak** and **Sarah Cormier**

Outline

Outline

- Non-definitions of stiffness

Outline

- Non-definitions of stiffness
- Practical stiffness detection

Outline

- Non-definitions of stiffness
- Practical stiffness detection
- Type-insensitive software

Outline

- Non-definitions of stiffness
- Practical stiffness detection
- Type-insensitive software
- Experiments

Outline

- Non-definitions of stiffness
- Practical stiffness detection
- Type-insensitive software
- Experiments
- Conclusions and future work

Non-definitions of stiffness

We are interested in the *efficient numerical solution* of initial-value problems (IVPs) for ordinary differential equations (ODEs):

Non-definitions of stiffness

We are interested in the *efficient numerical solution* of initial-value problems (IVPs) for ordinary differential equations (ODEs):

$$\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}), \quad \mathbf{y}(0) = \mathbf{y}_0, \quad t = [0, t_f].$$

Non-definitions of stiffness

In terms of *cost per step*,
explicit methods are the most efficient.

Non-definitions of stiffness

Here is the algorithm for the simplest possible explicit method (Euler's method) with constant time step:

Non-definitions of stiffness

Here is the algorithm for the simplest possible explicit method (Euler's method) with constant time step:

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \Delta t \mathbf{f}(t_n, \mathbf{y}_n), \quad n = 0, 1, 2, \dots$$

Non-definitions of stiffness

Here is the algorithm for the simplest possible explicit method (Euler's method) with constant time step:

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \Delta t \mathbf{f}(t_n, \mathbf{y}_n), \quad n = 0, 1, 2, \dots$$

→ an **explicit** formula to advance the numerical solution $\mathbf{y}_n \approx \mathbf{y}(t_n)$ at time $t = t_n$ to $\mathbf{y}_{n+1} \approx \mathbf{y}(t_{n+1})$ at time $t = t_{n+1} = t + \Delta t$.

Explicit Runge-Kutta methods

Explicit Runge-Kutta methods

for $i = 1, \dots, s$

$$\mathbf{K}_i = \mathbf{f} \left(t_n + c_i \Delta t, \mathbf{y}_n + \sum_{j=1}^{i-1} a_{ij} \mathbf{K}_j \right)$$

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \Delta t \sum_{i=1}^s b_i \mathbf{K}_i$$

General Runge-Kutta methods

In general Runge-Kutta methods are completely characterized by the coefficients a_{ij} , b_j , c_j .

General Runge-Kutta methods

In general Runge-Kutta methods are completely characterized by the coefficients a_{ij} , b_j , c_j .

They can be neatly summarized by means of the *Butcher tableau*:

$$\begin{array}{c|c} \mathbf{c} & \mathbf{A} \\ \hline & \mathbf{b}^T \end{array},$$

where $\mathbf{c}, \mathbf{b} \in \mathbb{R}^s$, $\mathbf{A} \in \mathbb{R}^{s \times s}$, and $\sum_{j=1}^s a_{ij} = c_i$, $i = 1, 2, \dots, s$.

General Runge-Kutta methods

In general Runge-Kutta methods are completely characterized by the coefficients a_{ij} , b_j , c_j .

They can be neatly summarized by means of the *Butcher tableau*:

$$\begin{array}{c|c} \mathbf{c} & \mathbf{A} \\ \hline & \mathbf{b}^T \end{array},$$

where $\mathbf{c}, \mathbf{b} \in \mathbb{R}^s$, $\mathbf{A} \in \mathbb{R}^{s \times s}$, and

$$\sum_{j=1}^s a_{ij} = c_i, \quad i = 1, 2, \dots, s.$$

For ERKs, A is *strictly lower-triangular*.

General Runge-Kutta methods

In general Runge-Kutta methods are completely characterized by the coefficients a_{ij} , b_j , c_j .

They can be neatly summarized by means of the *Butcher tableau*:

$$\begin{array}{c|c} \mathbf{c} & \mathbf{A} \\ \hline & \mathbf{b}^T \end{array},$$

where $\mathbf{c}, \mathbf{b} \in \mathbb{R}^s$, $\mathbf{A} \in \mathbb{R}^{s \times s}$, and

$$\sum_{j=1}^s a_{ij} = c_i, \quad i = 1, 2, \dots, s.$$

For ERKs, A is *strictly lower-triangular*.

Easy to program! What's the catch?

Non-definitions of stiffness

Consider the following (famous) *test problem*:

$$\dot{y} = \lambda y, \quad y(0) = 1.$$

Exact solution: $y(t) = e^{\lambda t}$

Non-definitions of stiffness

Consider the following (famous) *test problem*:

$$\dot{y} = \lambda y, \quad y(0) = 1.$$

Exact solution: $y(t) = e^{\lambda t}$

What is the numerical solution using Euler's method?

Non-definitions of stiffness

Consider the following (famous) *test problem*:

$$\dot{y} = \lambda y, \quad y(0) = 1.$$

Exact solution: $y(t) = e^{\lambda t}$

What is the numerical solution using Euler's method?

$$\begin{aligned} y_{n+1} &= y_n + \Delta t f(t_n, y_n) \\ &= y_n + \Delta t \lambda y_n \\ &= (1 + \Delta t \lambda) y_n = (1 + \Delta t \lambda)^n y_0 \end{aligned}$$

Non-definitions of stiffness

Recall:

$$y(t) = e^{\lambda t}, \quad y_{n+1} = (1 + \Delta t \lambda)^n y_0.$$

So if $\Re(\lambda) < 0$ but $|(1 + \Delta t \lambda)| > 1$, then

$$|y(t)| \rightarrow 0 \quad \text{but} \quad |y_n| \rightarrow \infty \quad \text{as} \quad n \rightarrow \infty!$$

This is called *numerical instability*.

Non-definitions of stiffness

Recall:

$$y(t) = e^{\lambda t}, \quad y_{n+1} = (1 + \Delta t \lambda)^n y_0.$$

So if $\Re(\lambda) < 0$ but $|(1 + \Delta t \lambda)| > 1$, then

$$|y(t)| \rightarrow 0 \quad \text{but} \quad |y_n| \rightarrow \infty \quad \text{as} \quad n \rightarrow \infty!$$

This is called *numerical instability*.

For this not to happen, Δt must be small . . .

Non-definitions of stiffness

Recall:

$$y(t) = e^{\lambda t}, \quad y_{n+1} = (1 + \Delta t \lambda)^n y_0.$$

So if $\Re(\lambda) < 0$ but $|(1 + \Delta t \lambda)| > 1$, then

$$|y(t)| \rightarrow 0 \quad \text{but} \quad |y_n| \rightarrow \infty \quad \text{as} \quad n \rightarrow \infty!$$

This is called *numerical instability*.

For this not to happen, Δt must be small . . .

In practice Δt must be so small that the overall computation becomes **inefficient**.

Non-definitions of stiffness

Recall:

$$y(t) = e^{\lambda t}, \quad y_{n+1} = (1 + \Delta t \lambda)^n y_0.$$

So if $\Re(\lambda) < 0$ but $|(1 + \Delta t \lambda)| > 1$, then

$$|y(t)| \rightarrow 0 \quad \text{but} \quad |y_n| \rightarrow \infty \quad \text{as} \quad n \rightarrow \infty!$$

This is called *numerical instability*.

For this not to happen, Δt must be small ...

In practice Δt must be so small that the overall computation becomes **inefficient**.

This is a non-definition of stiffness.

Non-definitions of stiffness

Recalling the test problem:

$$\dot{y} = \lambda y, \quad y(0) = 1.$$

Exact solution: $y(t) = e^{\lambda t}$.

Consider the **implicit Euler method**:

Non-definitions of stiffness

Recalling the test problem:

$$\dot{y} = \lambda y, \quad y(0) = 1.$$

Exact solution: $y(t) = e^{\lambda t}$.

Consider the **implicit Euler method**:

$$\begin{aligned} y_{n+1} &= y_n + \Delta t f(t_{n+1}, y_{n+1}) \\ &= y_n + \Delta t \lambda y_{n+1} \\ &= \frac{y_n}{1 - \Delta t \lambda} = \frac{y_0}{(1 - \Delta t \lambda)^n} \end{aligned}$$

Non-definitions of stiffness

Recall:

$$y(t) = e^{\lambda t}, \quad y_{n+1} = \frac{1}{(1 - \Delta t \lambda)^n} y_0.$$

So now if $\Re(\lambda) < 0$, then

$$y(t) \rightarrow 0, \quad |y_n| \rightarrow 0 \quad \text{as} \quad n \rightarrow \infty \quad \text{for all } \Delta t!$$

Non-definitions of stiffness

Recall:

$$y(t) = e^{\lambda t}, \quad y_{n+1} = \frac{1}{(1 - \Delta t \lambda)^n} y_0.$$

So now if $\Re(\lambda) < 0$, then

$$y(t) \rightarrow 0, \quad |y_n| \rightarrow 0 \quad \text{as} \quad n \rightarrow \infty \quad \text{for all } \Delta t!$$

Thus Δt need not be chosen on the basis of stability.

Non-definitions of stiffness

Recall:

$$y(t) = e^{\lambda t}, \quad y_{n+1} = \frac{1}{(1 - \Delta t \lambda)^n} y_0.$$

So now if $\Re(\lambda) < 0$, then

$$y(t) \rightarrow 0, \quad |y_n| \rightarrow 0 \quad \text{as} \quad n \rightarrow \infty \quad \text{for all } \Delta t!$$

Thus Δt need not be chosen on the basis of stability.

i.e., Δt can be chosen on the basis of accuracy.

Non-definitions of stiffness

Recall:

$$y(t) = e^{\lambda t}, \quad y_{n+1} = \frac{1}{(1 - \Delta t \lambda)^n} y_0.$$

So now if $\Re(\lambda) < 0$, then

$$y(t) \rightarrow 0, \quad |y_n| \rightarrow 0 \quad \text{as} \quad n \rightarrow \infty \quad \text{for all } \Delta t!$$

Thus Δt need not be chosen on the basis of stability.

i.e., Δt can be chosen on the basis of accuracy.

This is one non-definition of stiffness.

Non-definitions of stiffness

What's the catch? (Why not always use implicit methods?)

Implicit methods generally involve *solving a (nonlinear) system of equations at each step.*

This only pays if increase in (acceptable) Δt more than offsets increased cost per step.

(It is also hard to write general purpose software!)

Non-definitions of stiffness

Non-definitions of stiffness

Stiffness is about efficiency,
so pragmatic definitions have more potential.

Non-definitions of stiffness

Stiffness is about efficiency,
so pragmatic definitions have more potential.

- “... a precise [mathematical] definition of stiffness is not crucial for practical purposes.”
[Ekeland, Owren, Øines; ACM TOMS, 1998]

Non-definitions of stiffness

Stiffness is about efficiency,
so pragmatic definitions have more potential.

- “... a precise [mathematical] definition of stiffness is not crucial for practical purposes.” [Ekeland, Owren, Øines; ACM TOMS, 1998]
- “Stiff equations are problems for which explicit methods don’t work.” [Hairer and Wanner]

Non-definitions of stiffness

Stiffness is about efficiency,
so pragmatic definitions have more potential.

- “... a precise [mathematical] definition of stiffness is not crucial for practical purposes.”
[Ekeland, Owren, Øines; ACM TOMS, 1998]
- “Stiff equations are problems for which explicit methods don’t work *as well as implicit methods*.”

Non-definitions of stiffness

We are now ready to “define” stiffness.

Non-definitions of stiffness

We are now ready to “define” stiffness.

For a given IVP (including interval $[0, t_f]$), order of discretization, and error tolerance,

Non-definitions of stiffness

We are now ready to “define” stiffness.

For a given IVP (including interval $[0, t_f]$), order of discretization, and error tolerance,

*on the sub-intervals where it is more efficient to use an implicit method than an explicit one, we say the problem is **stiff**.*

Non-definitions of stiffness

We are now ready to “define” stiffness.

For a **given IVP (including interval $[0, t_f]$)**, **order of discretization**, and **error tolerance**,

*on the sub-intervals where it is more efficient to use an implicit method than an explicit one, we say the problem is **stiff**.*

Terminology first used in print by Curtiss and Hirschfelder [PNAS, 1952].

Non-definitions of stiffness

We are now ready to “define” stiffness.

For a **given IVP (including interval $[0, t_f]$)**, **order of discretization**, and **error tolerance**,

*on the sub-intervals where it is more efficient to use an implicit method than an explicit one, we say the problem is **stiff**.*

Terminology first used in print by Curtiss and Hirschfelder [PNAS, 1952].

Dissociate stiffness from ODE without context.

Non-definitions of stiffness

“Unintuitive”(?) conclusions of this definition:

Non-definitions of stiffness

“Unintuitive”(?) conclusions of this definition:

- *Stiffness is local* (not usually global).

Non-definitions of stiffness

“Unintuitive”(?) conclusions of this definition:

- *Stiffness is local* (not usually global).
- In principle, *no problem is stiff* if the error tolerance is small enough.

Non-definitions of stiffness

“Unintuitive”(?) conclusions of this definition:

- *Stiffness is local* (not usually global).
- In principle, *no problem is stiff* if the error tolerance is small enough.
- If the problem is stiff, an explicit method will yield a numerical solution that is *too accurate*.

Non-definitions of stiffness

Stiffness in the numerical solution of ODEs is

Non-definitions of stiffness

Stiffness in the numerical solution of ODEs is

- subtle

Non-definitions of stiffness

Stiffness in the numerical solution of ODEs is

- subtle
- pervasive

Non-definitions of stiffness

Stiffness in the numerical solution of ODEs is

- subtle
- pervasive
- important

Non-definitions of stiffness

Stiffness in the numerical solution of IVPs is *subtle*:

- It's not just about eigenvalues (even for linear systems).
- It's not just about the distribution of eigenvalues.
- It's not just about the behaviour of neighbouring solutions.
- It's not just about mathematics.

Non-definitions of stiffness

Stiffness in the numerical solution of IVPs is *pervasive*:

- problems with components varying on different time scales
- method-of-lines discretizations of partial differential equations

Non-definitions of stiffness

Stiffness in the numerical solution of IVPs is *important*:

- It's an efficiency thing.
- If we can detect stiffness (and non-stiffness) reliably, we can “always” use the most appropriate methods.

Appropriate handling of stiffness determine whether a solution can be obtained or not.

Practical stiffness detection

Stiffness detection comprises of two important aspects:

Practical stiffness detection

Stiffness detection comprises of two important aspects:

- *when* to check for stiffness

Practical stiffness detection

Stiffness detection comprises of two important aspects:

- *when* to check for stiffness
- *how* to check for stiffness

When to check for stiffness

When to check for stiffness

- Ideally, check at every step.

When to check for stiffness

- Ideally, check at every step.
- But this requires a cheap test; cheap tests are unreliable!

When to check for stiffness

- Ideally, check at every step.
- But this requires a cheap test; cheap tests are unreliable!
- Instead, **use a reliable test periodically.**

When to check for stiffness

When to check for stiffness

- Every so often (e.g., every MAXFCN right-hand side evaluations).

When to check for stiffness

- Every so often (e.g., every MAXFCN right-hand side evaluations).
- If there are at least 10 step rejections in last 50 successful steps and

$$\Delta t \in \left[\frac{\Delta t_{\text{ave}}}{a_{\text{max}}}, a_{\text{max}} \Delta t_{\text{ave}} \right],$$

where $a_{\text{max}} = 5$ (i.e., stepsize fairly constant).

When to check for stiffness

- Every so often (e.g., every MAXFCN right-hand side evaluations).
- If there are at least 10 step rejections in last 50 successful steps and

$$\Delta t \in \left[\frac{\Delta t_{\text{ave}}}{a_{\text{max}}}, a_{\text{max}} \Delta t_{\text{ave}} \right],$$

where $a_{\text{max}} = 5$ (i.e., stepsize fairly constant).

Note: If completion predicted soon, just do it!

How to check for stiffness

How to check for stiffness

- Constant-stepsize method [Shampine, 1975]

How to check for stiffness

- Constant-stepsize method [Shampine, 1975]
- Low-order comparison formulas [Shampine, 1984]

How to check for stiffness

- Constant-stepsize method [Shampine, 1975]
- Low-order comparison formulas [Shampine, 1984]
- Lipschitz-constant method [Shampine, 1979]

How to check for stiffness

- Constant-stepsize method [Shampine, 1975]
- Low-order comparison formulas [Shampine, 1984]
- Lipschitz-constant method [Shampine, 1979]
- Arnoldi-based method [EOØ, 1998]

How to check for stiffness

- Constant-stepsize method [Shampine, 1975]
- Low-order comparison formulas [Shampine, 1984]
- Lipschitz-constant method [Shampine, 1979]
- Arnoldi-based method [EOØ, 1998]
- Problem time per unit real time

How to check for stiffness

More details on the Lipschitz-constant method:

How to check for stiffness

More details on the Lipschitz-constant method:

- Recall: $\|\mathbf{f}(t, \mathbf{y}) - \mathbf{f}(t, \tilde{\mathbf{y}})\| \leq L\|\mathbf{y} - \tilde{\mathbf{y}}\|$.

How to check for stiffness

More details on the Lipschitz-constant method:

- Recall: $\|\mathbf{f}(t, \mathbf{y}) - \mathbf{f}(t, \tilde{\mathbf{y}})\| \leq L\|\mathbf{y} - \tilde{\mathbf{y}}\|$.
- Recall: $L = \sup \|\mathbf{J}\|$, where $\mathbf{J} = \partial\mathbf{f}/\partial\mathbf{y}$.

How to check for stiffness

More details on the Lipschitz-constant method:

- Recall: $\|\mathbf{f}(t, \mathbf{y}) - \mathbf{f}(t, \tilde{\mathbf{y}})\| \leq L\|\mathbf{y} - \tilde{\mathbf{y}}\|$.
- Recall: $L = \sup \|\mathbf{J}\|$, where $\mathbf{J} = \partial\mathbf{f}/\partial\mathbf{y}$.
- Relies on the fact $\rho(\mathbf{J}) \leq \|\mathbf{J}\| \leq L$.

How to check for stiffness

More details on the Lipschitz-constant method:

- Recall: $\|\mathbf{f}(t, \mathbf{y}) - \mathbf{f}(t, \tilde{\mathbf{y}})\| \leq L\|\mathbf{y} - \tilde{\mathbf{y}}\|$.
- Recall: $L = \sup \|\mathbf{J}\|$, where $\mathbf{J} = \partial\mathbf{f}/\partial\mathbf{y}$.
- Relies on the fact $\rho(\mathbf{J}) \leq \|\mathbf{J}\| \leq L$.
- For Dormand-Prince 5(4),

$$L_{\text{est}} = \frac{\|\mathbf{K}_6 - \mathbf{K}_7\|}{\|\mathbf{y}_6 - \mathbf{y}_7\|} \approx \rho(\mathbf{J}).$$

Type-insensitive software

First, a note about odeToJava, the software used to produce the results:

- available at
`www.netlib.org/ode/odeToJava.tgz`

Type-insensitive software

First, a note about odeToJava, the software used to produce the results:

- available at
`www.netlib.org/ode/odeToJava.tgz`
- functionality includes constant-stepsize, error/stepsize control, interpolation, event location, stiffness detection

Type-insensitive software

First, a note about `odeToJava`, the software used to produce the results:

- available at `www.netlib.org/ode/odeToJava.tgz`
- functionality includes constant-stepsize, error/stepsize control, interpolation, event location, stiffness detection
- linearly implicit IMEX Runge-Kutta methods

Type-insensitive software

First, a note about odeToJava, the software used to produce the results:

- available at www.netlib.org/ode/odeToJava.tgz
- functionality includes constant-stepsize, error/stepsize control, interpolation, event location, stiffness detection
- linearly implicit IMEX Runge-Kutta methods
- plotter

Type-insensitive software

Algorithm for a new type-insensitive code:

1. Start with explicit method.
2. Use *when-to-check* heuristics.
3. If stiffness suspected, determine relative efficiencies (problem time per unit real time); use the more efficient method:
 - (a) If explicit, return to 1.
 - (b) If implicit, continue; try explicit periodically (*non-stiffness detection*).

Type-insensitive software

Results for the Non-stiff DE Test Set
[Hull, Enright, Fellen, Sedgwick; SINUM, 1972]

- A: Single equations
- B: Small systems
- C: Moderate systems
- D: Orbit equations
- E: Higher-order equations

Non-stiff DE Test Set

- A: Single equations (execution time in ms)

	DoPr 5(4)	ARK 5(4)	Type-insensitive
A1	16	16	16
A2	9	12	9
A3	16	31	16
A4	8	9	8
A5	7	11	7

Non-stiff DE Test Set

- B: Small systems (execution time in ms)

	DoPr 5(4)	ARK 5(4)	Type-insensitive
B1	32	64	32
B2	29	29	29
B3	29	39	29
B4	35	30	35
B5	27	64	27

Non-stiff DE Test Set

- C: Moderate systems (execution time in ms)

	DoPr 5(4)	ARK 5(4)	Type-insensitive
C1	87	150	87
C2	261	174	257
C3	97	134	98
C4	699	2986	697
C5	28	64	28

Non-stiff DE Test Set

- D: Orbit equations (execution time in ms)

	DoPr 5(4)	ARK 5(4)	Type-insensitive
D1	38	98	40
D2	40	114	43
D3	49	136	50
D4	59	167	59
D5	81	225	79

Non-stiff DE Test Set

- E: Higher-order equations (time in ms)

	DoPr 5(4)	ARK 5(4)	Type-insensitive
E1	18	26	19
E2	43	78	43
E3	36	90	37
E4	12	16	12
E5	12	21	12

Type-insensitive software

Results for the Stiff DE Test Set [Enright, Hull, Linberg; BIT, 1975]

- A: Linear with real eigenvalues
- B: Linear with non-real eigenvalues
- C: Non-linear coupling
- D: Non-linear with real eigenvalues

Stiff DE Test Set

- A: Linear with real eigenvalues
(execution time in ms)

	DoPr 5(4)	ARK 5(4)	Type-insensitive
A1	1348	51	1338
A2*	130511	141	4870
A3*	61645	59	1774
A4*	140970	300	8802

Stiff DE Test Set

- B: Linear with non-real eigenvalues (execution time in ms)

	DoPr 5(4)	ARK 5(4)	Type-insensitive
B1*	1813	234	275
B2	184	81	186
B3	226	88	225
B4	450	122	444
B5	1973	250	1930

Stiff DE Test Set

- C: Non-linear coupling (execution time in ms)

	DoPr 5(4)	ARK 5(4)	Type-insensitive
C1	1079	60	1058
C2	589	55	570
C3	573	60	554
C4	574	70	562
C5	556	75	542

Stiff DE Test Set

- D: Non-linear with real eigenvalues (execution time in ms)

	DoPr 5(4)	ARK 5(4)	Type-insensitive
D1*	3304	224	1534
D2*	66932	1365	1357
D3	1607	78	1565
D4*	63059	460	424
D5*	8830	4020	10637

Stiff DE Test Set

- E: Non-linear with non-real eigenvalues
(execution time in ms)

	DoPr 5(4)	ARK 5(4)	Type-insensitive
E1	122	43	126
E2	11	16	11
E3*	8965	329	1502
E4*	655632	537	3159
E5*	—	—	—

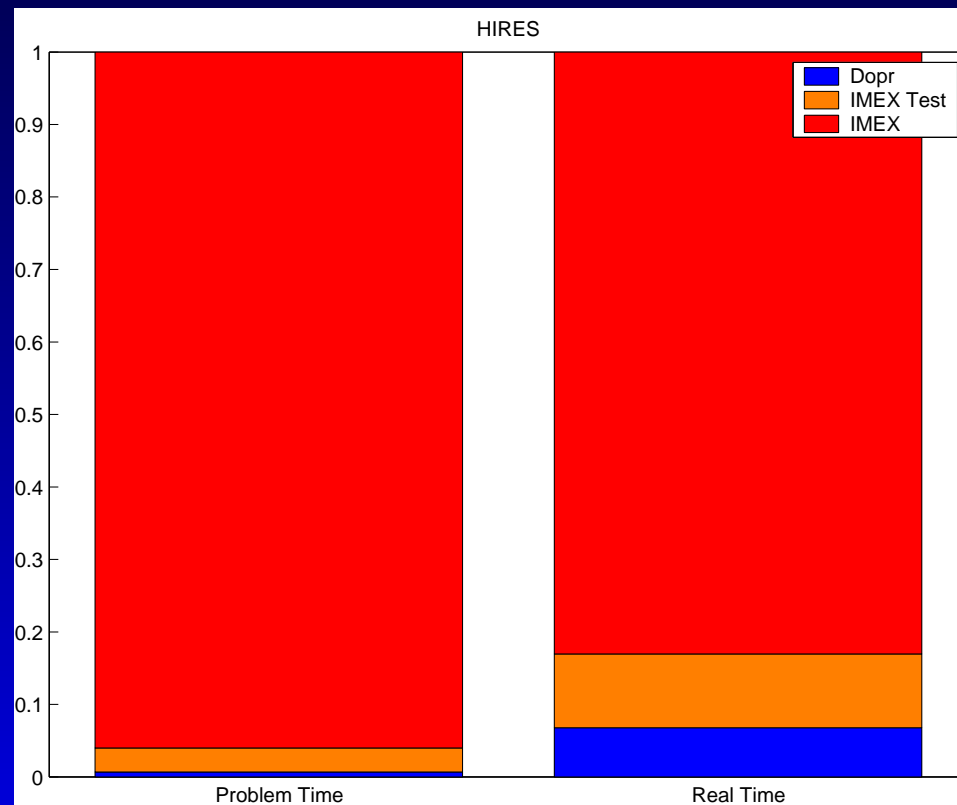
Stiff DE Test Set

- F: Chemical Kinetics (execution time in ms)

	DoPr 5(4)	ARK 5(4)	Type-insensitive
F1	–	–	–
F2	200	23	200
F3*	814119	78	105
F4*	153559	82	1870

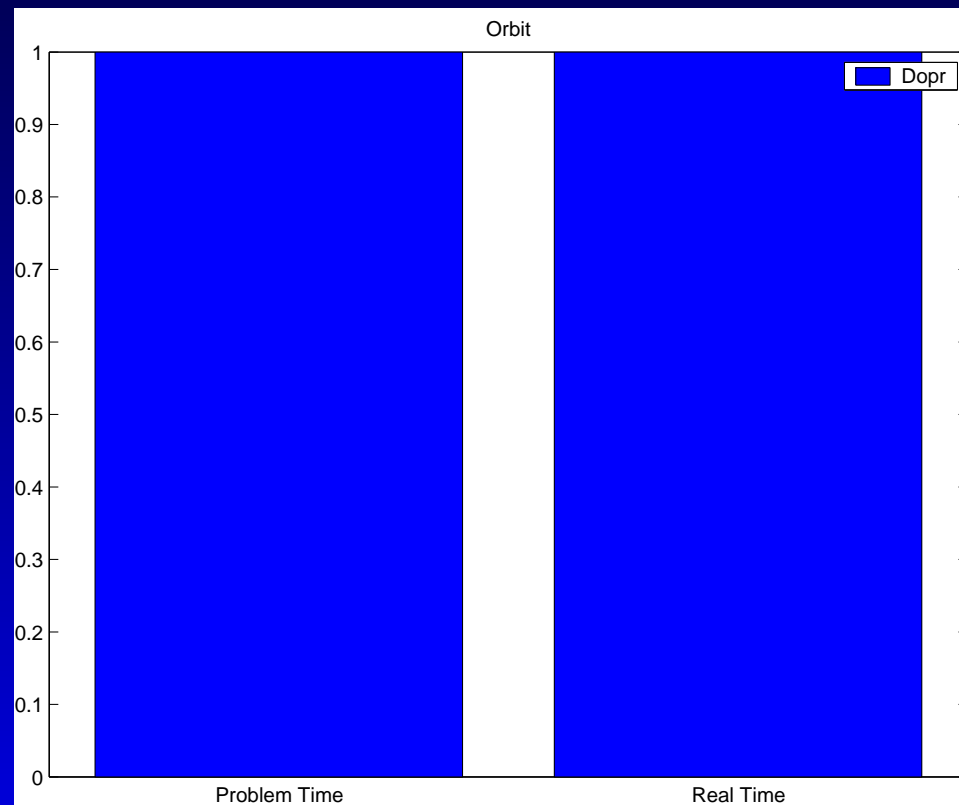
Experiments: HIRES*

DoPr 5(4)	ARK 5(4)	Type-insensitive
26820	1919	1823



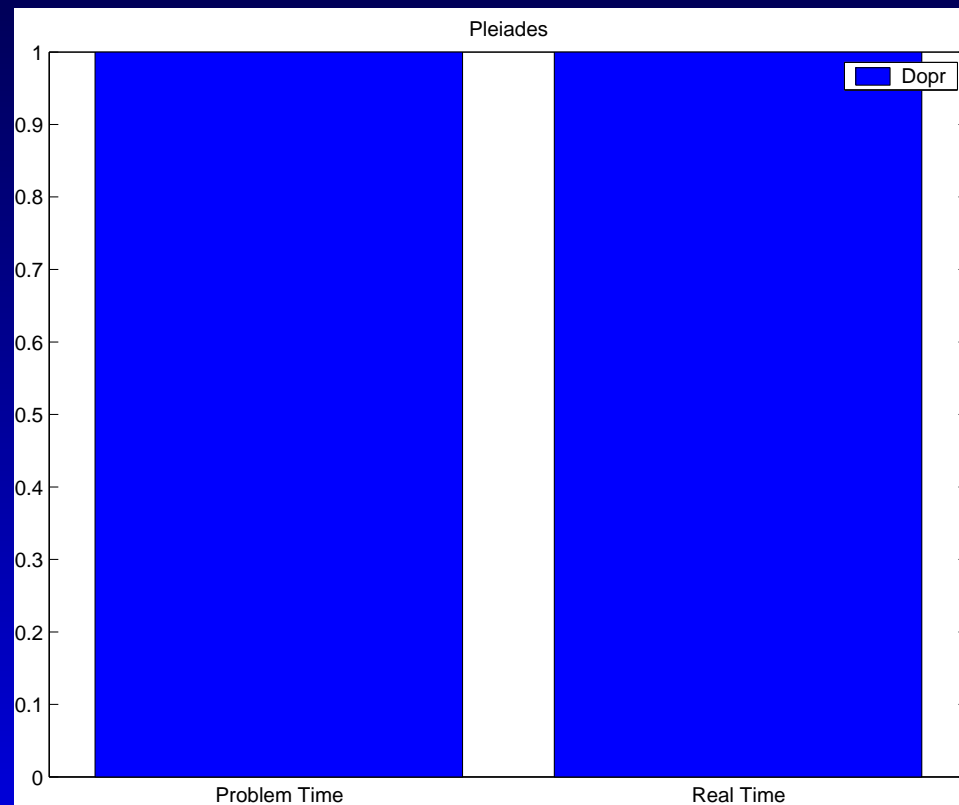
Experiments: Orbit

DoPr 5(4)	ARK 5(4)	Type-insensitive
107	170	105



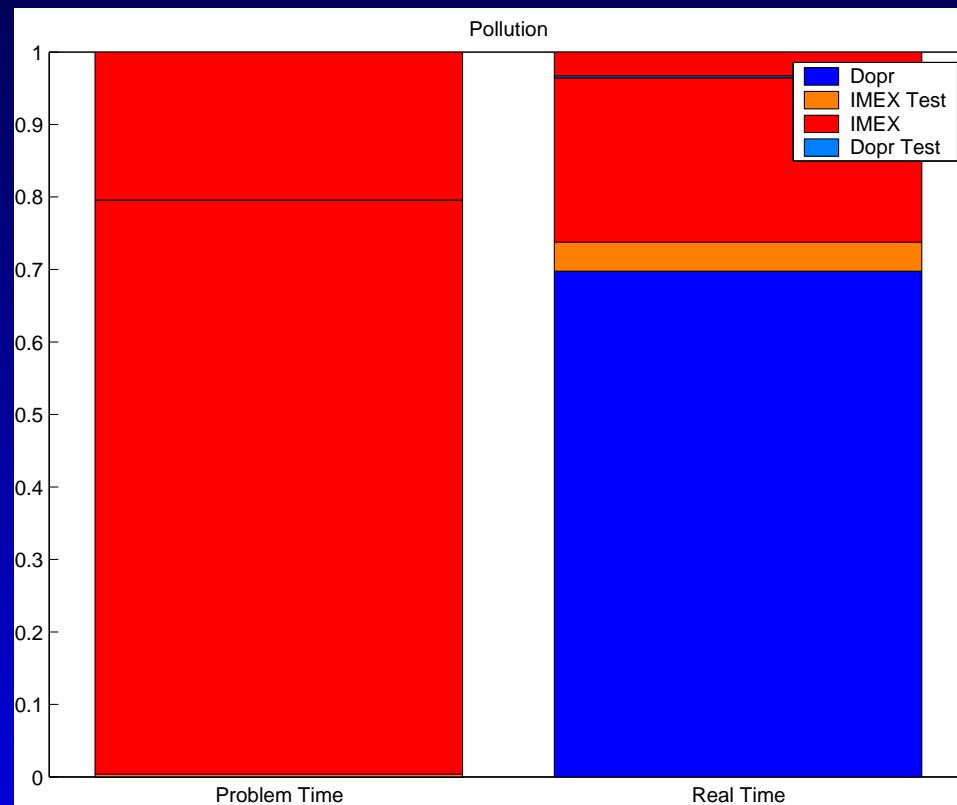
Experiments: Pleiades

DoPr 5(4)	ARK 5(4)	Type-insensitive
483	4757	473



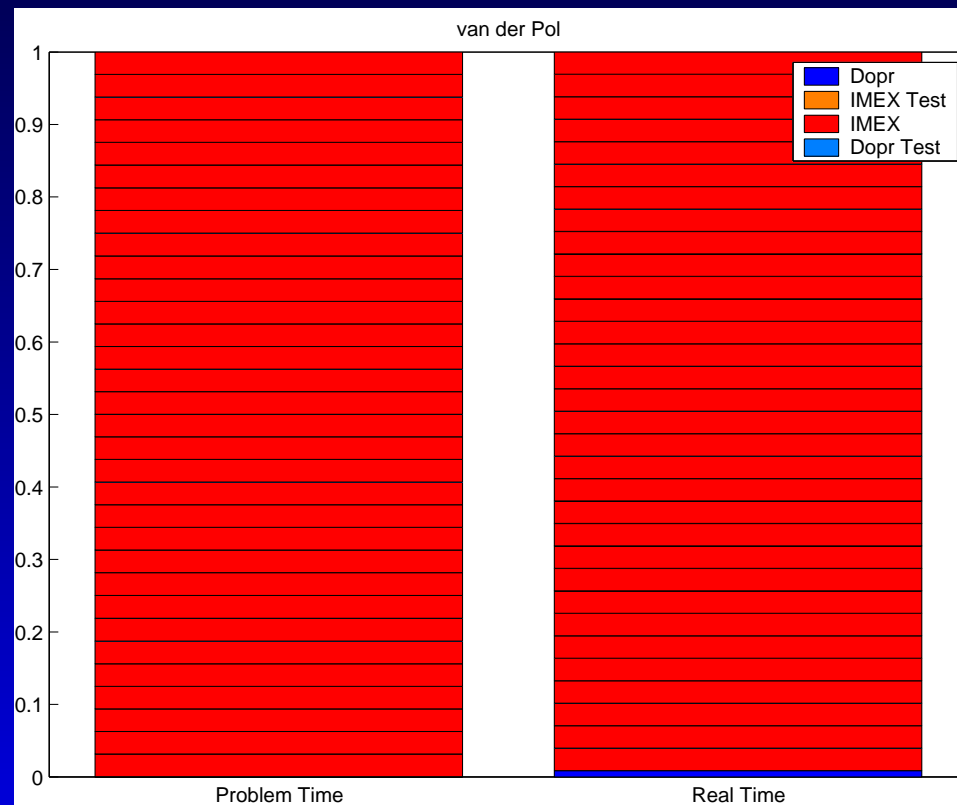
Experiments: Pollution*

DoPr 5(4)	ARK 5(4)	Type-insensitive
*	7085	22068



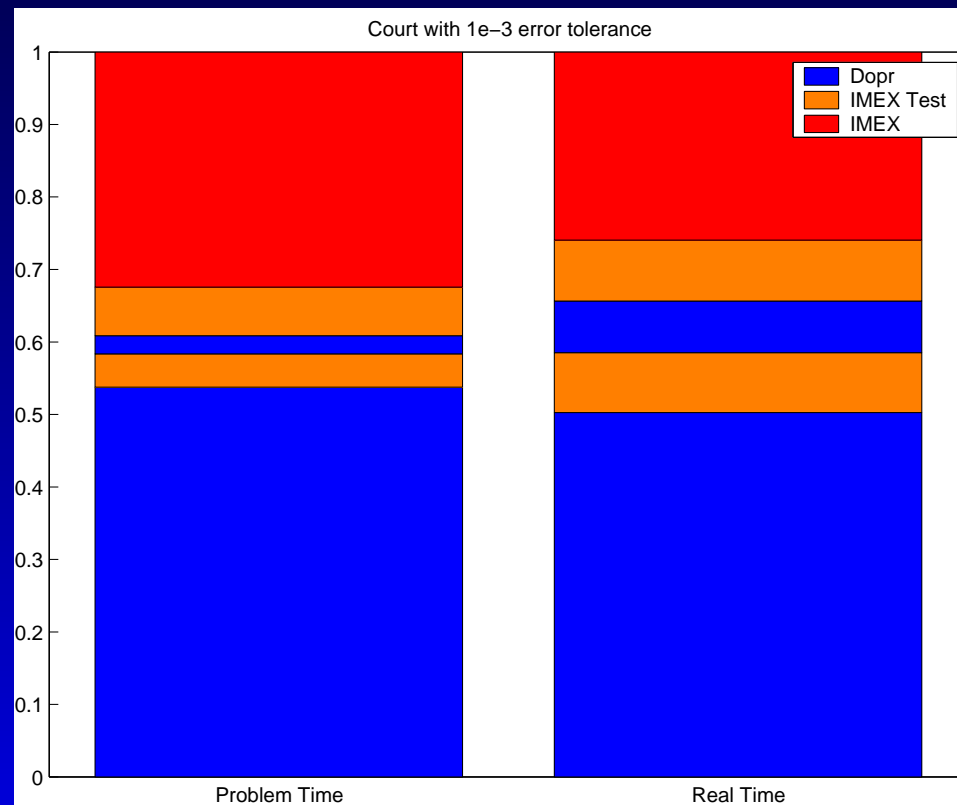
Experiments: van der Pol*

DoPr 5(4)	ARK 5(4)	Type-insensitive
*	158422	162241



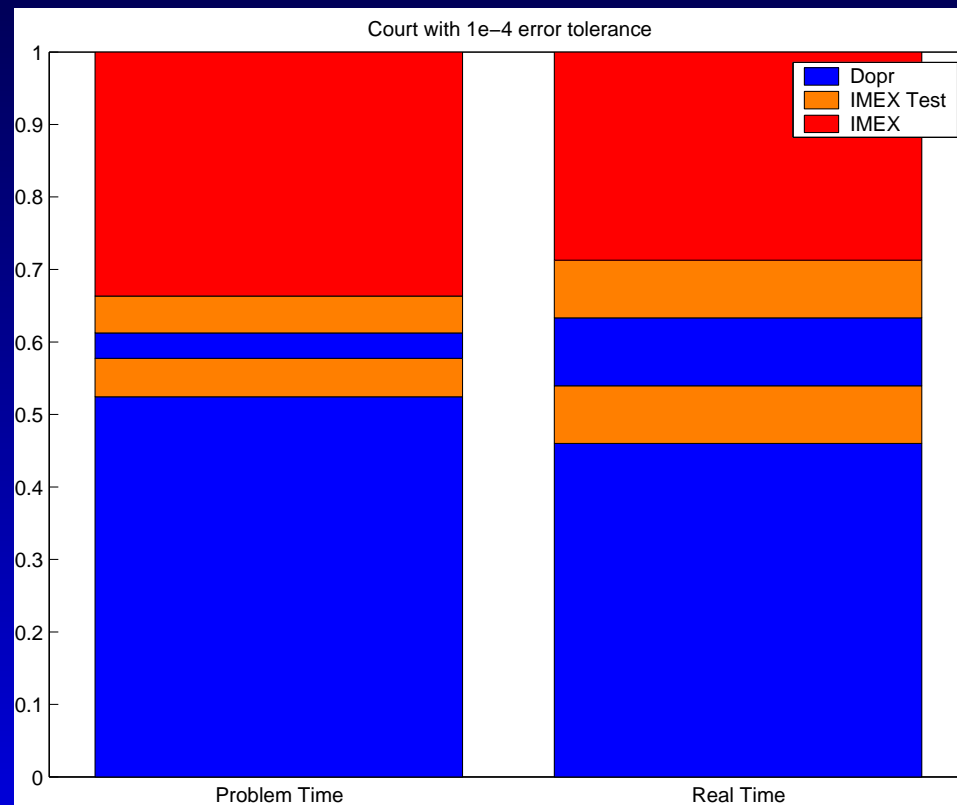
Experiments: Courtemanche

DoPr 5(4)	ARK 5(4)	Type-insensitive
35671	8592	11159



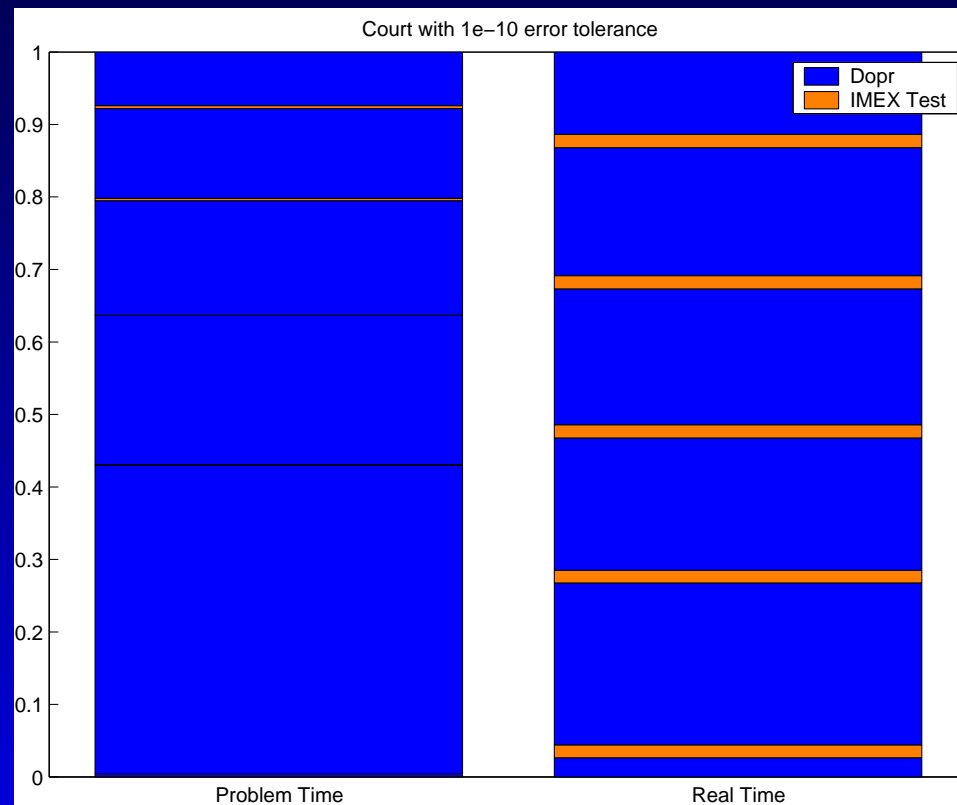
Experiments: Courtemanche

DoPr 5(4)	ARK 5(4)	Type-insensitive
35346	9368	11787



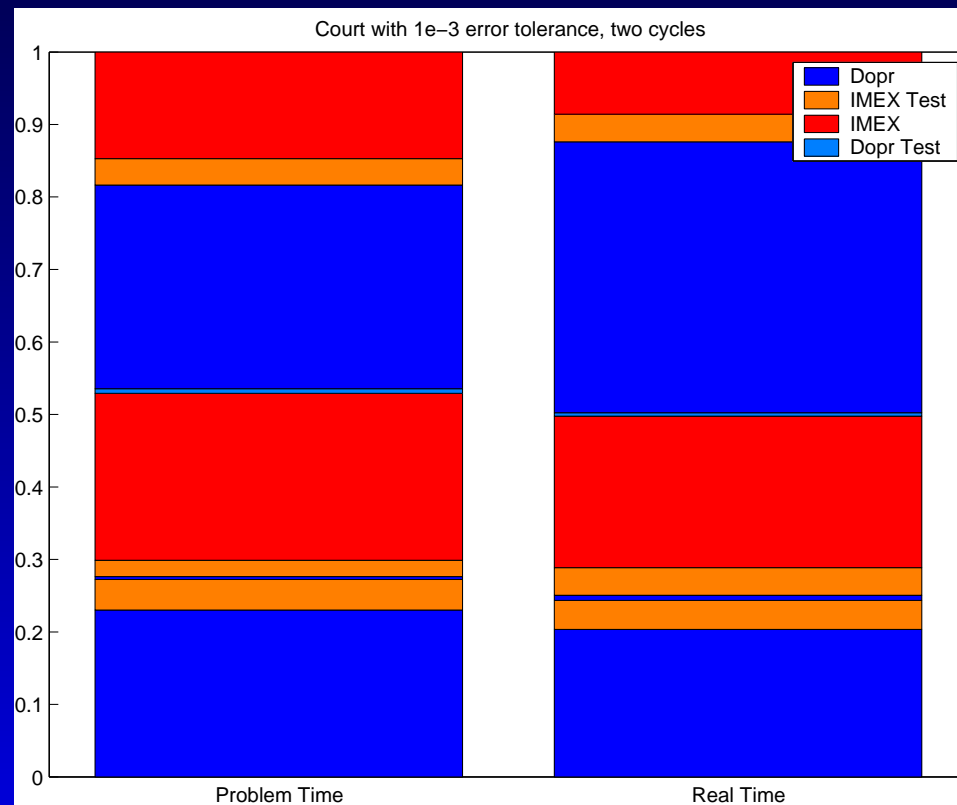
Experiments: Courtemanche

DoPr 5(4)	ARK 5(4)	Type-insensitive
49702	230982	52708



Experiments: Courtemanche

DoPr 5(4)	ARK 5(4)	Type-insensitive
75987	16909	23962



Conclusions and Future Work

Conclusions and Future Work

- Stiffness has many faces.

Conclusions and Future Work

- Stiffness has many faces.
- Stiffness is about efficiency.

Conclusions and Future Work

- Stiffness has many faces.
- Stiffness is about efficiency.
- Detecting non-stiffness is equally important!

Conclusions and Future Work

- Stiffness has many faces.
- Stiffness is about efficiency.
- Detecting non-stiffness is equally important!
- Promising design of type-insensitive code.