Solving differential equations in reduced- and mixed-precision

Matteo Croci

Applied Mathematics Seminar, Yale University, 9 March 2022

The University of Texas at Austin
Oden Institute for Computational Engineering and Sciences

Overview

1. Introduction and background

2. Solving parabolic PDEs in half precision (joint with M. B. Giles)

3. Mixed-precision explicit stabilised Runge-Kutta methods (with G. Rosilho De Souza)

4. Conclusions

Overview

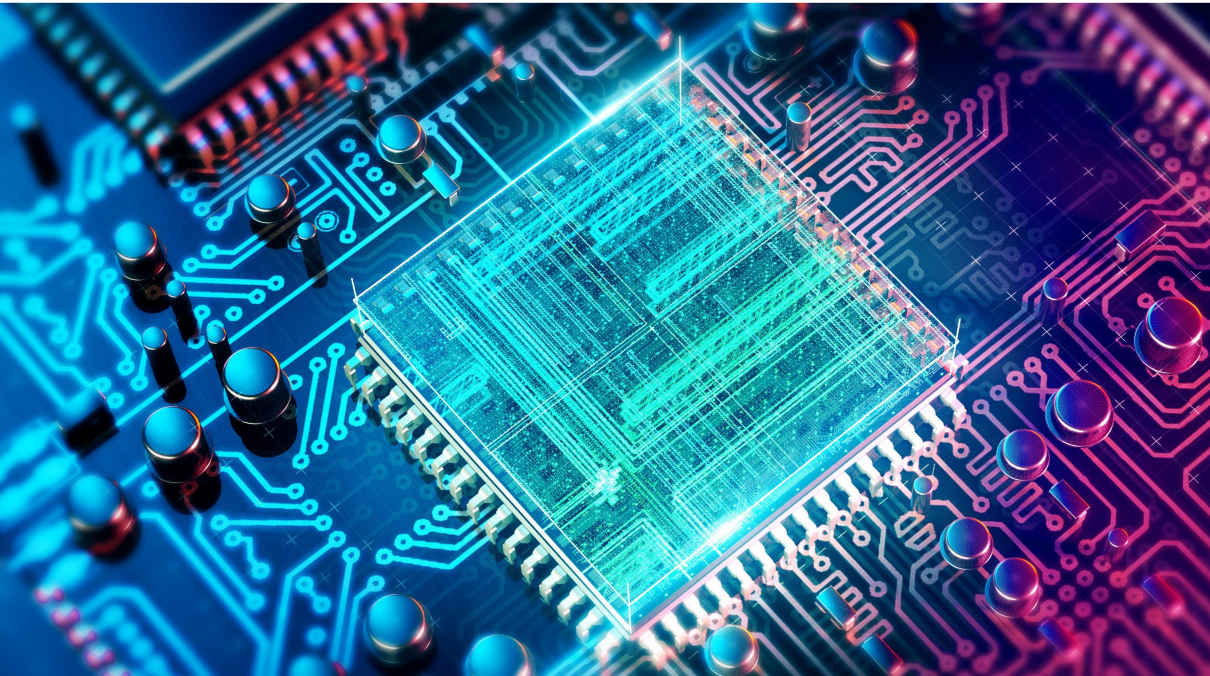
1. Introduction and background

2. Solving parabolic PDEs in half precision (joint with M. B. Giles)

3. Mixed-precision explicit stabilised Runge-Kutta methods (with G. Rosilho De Souza)

4. Conclusions

Objective: developing reduced- and mixed-precision DE solvers



Reduced- and mixed-precision algorithms[1]

Reduced-precision algorithms

Reduced-precision algorithms obtain an as accurate solution as possible given the precision while avoiding catastrophic rounding error accumulation.

[1] Review articles: [Abdelfattah et al. 2021], [Higham and Mary 2021], [C. et al. 2021].

Reduced- and mixed-precision algorithms[1]

Reduced-precision algorithms

Reduced-precision algorithms obtain an as accurate solution as possible given the precision while avoiding catastrophic rounding error accumulation.

Mixed-precision algorithms

Mixed-precision algorithms combine low- and high-precision computations in order to benefit from the performance gains of reduced-precision while retaining good accuracy.

[1] Review articles: [Abdelfattah et al. 2021], [Higham and Mary 2021], [C. et al. 2021].

Reduced- and mixed-precision algorithms[1]

Reduced-precision algorithms

Reduced-precision algorithms obtain an as accurate solution as possible given the precision while avoiding catastrophic rounding error accumulation.

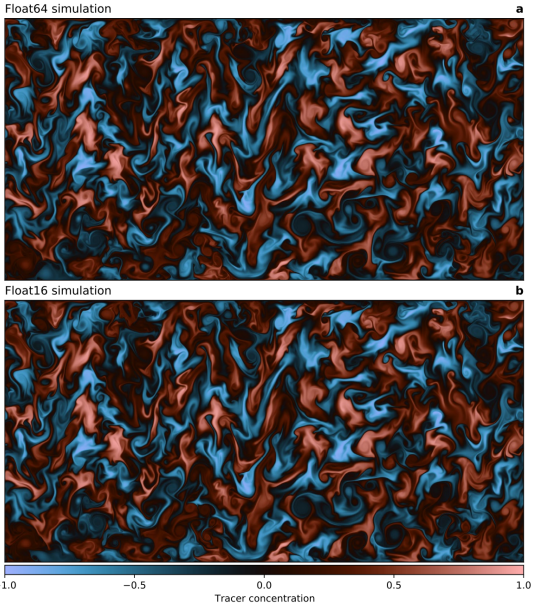
Mixed-precision algorithms

Mixed-precision algorithms combine low- and high-precision computations in order to benefit from the performance gains of reduced-precision while retaining good accuracy.

- This is now a very active field of investigation with many new developments led mainly by the numerical linear algebra community.
- Lots of new reduced-/mixed-precision algorithms: matrix factorizations, direct linear solvers, Krylov subspace methods, preconditioning, multigrid, nonlinear solvers.
- There is still much to discover on the topic.

[1] Review articles: [Abdelfattah et al. 2021], [Higham and Mary 2021], [C. et al. 2021].

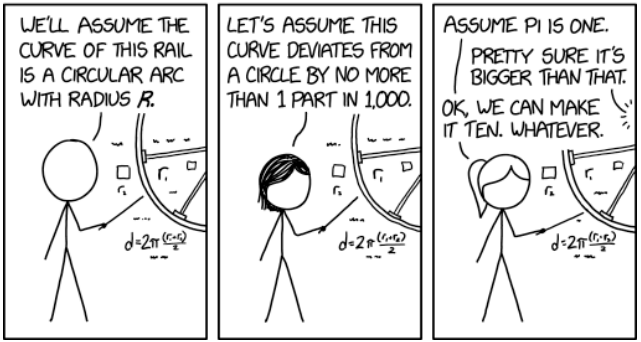
Nonlinear shallow water equations in half precision [Klöwer et al. 2021]



Performed on Fugaku (1st in the TOP500). All other results in this talk are emulated in software.

Floating point formats

Format	"Precision" u	$x_{\min}$	$x_{\max}$
bfloat16 (half)	$2^{-8} \approx 3.91 \times 10^{-3}$	1.18×10^{-38}	3.39×10^{38}
fp16 (half)	$2^{-11} \approx 4.88 \times 10^{-4}$	6.10×10^{-5}	6.55×10^{4}
fp32 (single)	$2^{-24} \approx 5.96 \times 10^{-8}$	1.18×10^{-38}	3.40×10^{38}
fp64 (double)	$2^{-53} \approx 1.11 \times 10^{-16}$	2.22×10^{-308}	1.80×10^{308}



exact arithmetic | half precision | no precision

Overview

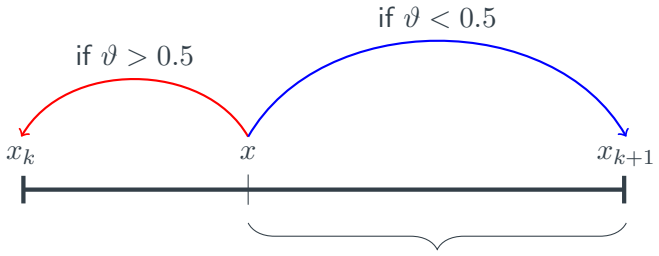
1. Introduction and background

2. Solving parabolic PDEs in half precision (joint with M. B. Giles)

3. Mixed-precision explicit stabilised Runge-Kutta methods (with G. Rosilho De Souza)

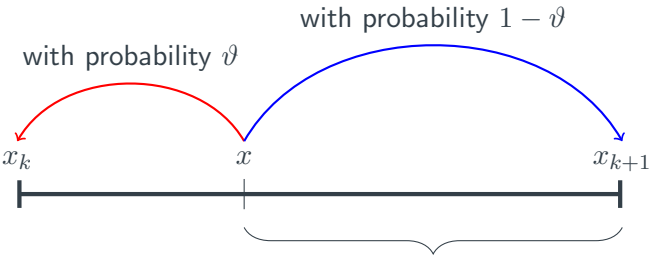
4. Conclusions

Round to nearest



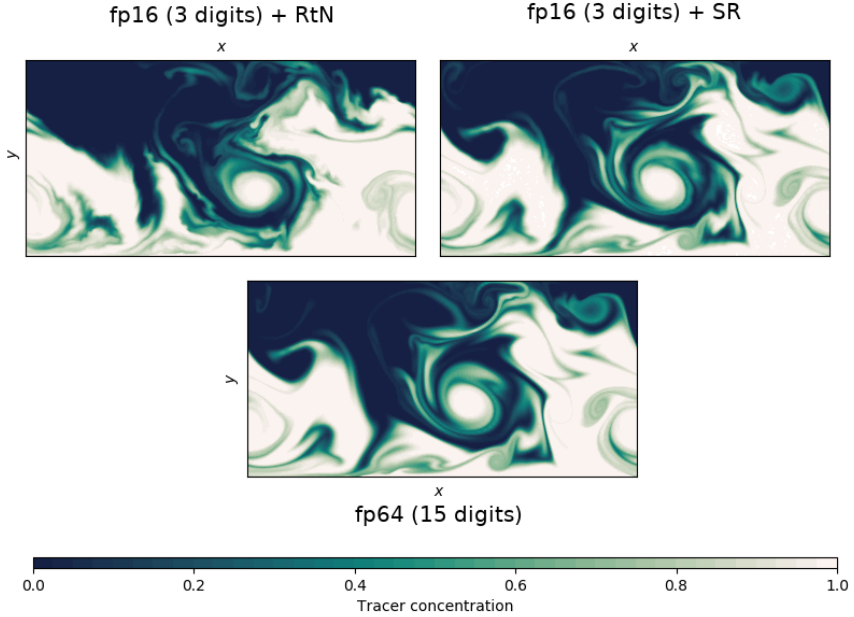
$$\mathsf{fl}(x) = x(1+\delta), \quad \text{with} \quad |\delta| \leq u.$$

Stochastic rounding



$\mathsf{sr}(x) = x(1+\delta(\omega)), \quad \text{with} \quad |\delta(\omega)| \leq 2u, \quad \text{and} \quad \mathbb{E}[sr(x)] = x.$

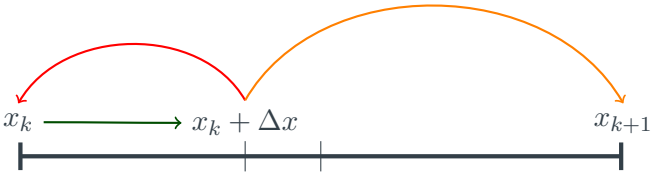
Interesting results (thanks to Milan Klöwer)



RtN might cause stagnation

RtN might cause stagnation

SR is resilient to stagnation



Other properties of SR [C. et al. 2022]

The good:

- SR roundoffs are zero mean and mean-independent: $\mathbb{E}[\delta_i|\delta_1, \ldots, \delta_{i-1}] = \mathbb{E}[\delta_i] = 0$.
- Rounding errors accumulate like $O(\sqrt{n}u)$ rather than $O(nu)$.
- Evaluation of linear functions is exact in expectation. $O(u^2)$ bias for C^2 functions.

The less good:

- Limited (yet growing) hardware support.

Heat equation as a test problem

$$\begin{cases} \dot{\mathrm{u}}(t,\boldsymbol{x}) = \nabla^2 \mathrm{u}(t,\boldsymbol{x}) + f(t,\boldsymbol{x}), & \boldsymbol{x} \in D = [0,1]^d, \quad t > 0, \\ \mathrm{u}(0,\boldsymbol{x}) = \mathrm{u}_0(x), & \boldsymbol{x} \in D, \\ \mathrm{u}(t,\boldsymbol{x}) = g(\boldsymbol{x}), & \boldsymbol{x} \in \partial D, \qquad t > 0. \end{cases}$$

Heat equation as a test problem

$$\begin{cases} \dot{\mathfrak{u}}(t,\boldsymbol{x}) = \nabla^2 \mathfrak{u}(t,\boldsymbol{x}) + f(t,\boldsymbol{x}), & \boldsymbol{x} \in D = [0,1]^d, \quad t > 0, \\ \mathfrak{u}(0,\boldsymbol{x}) = \mathfrak{u}_0(x), & \boldsymbol{x} \in D, \\ \mathfrak{u}(t,\boldsymbol{x}) = g(\boldsymbol{x}), & \boldsymbol{x} \in \partial D, \qquad\quad t > 0. \end{cases}$$

We use **finite differences** in space: let $\boldsymbol{U}(t) \in \mathbb{R}^K$ with $\boldsymbol{U}_i(t) \approx \mathfrak{u}(t,\boldsymbol{x}_i)$, then

$$\text{(e.g. in 1D)} \quad \dot{\boldsymbol{U}}_i = \frac{1}{h^2}\left(\boldsymbol{U}_{i+1} - 2\boldsymbol{U}_i + \boldsymbol{U}_{i-1}\right) + f(t,\boldsymbol{x}_i) \quad \rightsquigarrow \quad \dot{\boldsymbol{U}}(t) = -A\boldsymbol{U}(t) + \boldsymbol{F}(t).$$

In 1D, $A = \text{tridiag}(-1,2,-1)/h^2 \in \mathbb{R}^{K\times K}$. A is the (spd) **stiffness matrix** or discrete laplacian.

Heat equation as a test problem

$$\begin{cases} \dot{\mathfrak{u}}(t,\boldsymbol{x}) = \nabla^2 \mathfrak{u}(t,\boldsymbol{x}) + f(t,\boldsymbol{x}), & \boldsymbol{x} \in D = [0,1]^d, \quad t > 0, \\ \mathfrak{u}(0,\boldsymbol{x}) = \mathfrak{u}_0(x), & \boldsymbol{x} \in D, \\ \mathfrak{u}(t,\boldsymbol{x}) = g(\boldsymbol{x}), & \boldsymbol{x} \in \partial D, \qquad t > 0. \end{cases}$$

We use **finite differences** in space: let $\boldsymbol{U}(t) \in \mathbb{R}^K$ with $\boldsymbol{U}_i(t) \approx \mathfrak{u}(t,\boldsymbol{x}_i)$, then

$$\text{(e.g. in 1D)} \quad \dot{\boldsymbol{U}}_i = \frac{1}{h^2}\left(\boldsymbol{U}_{i+1} - 2\boldsymbol{U}_i + \boldsymbol{U}_{i-1}\right) + f(t,\boldsymbol{x}_i) \quad \rightsquigarrow \quad \dot{\boldsymbol{U}}(t) = -A\boldsymbol{U}(t) + \boldsymbol{F}(t).$$

In 1D, $A = \text{tridiag}(-1,2,-1)/h^2 \in \mathbb{R}^{K\times K}$. A is the (spd) **stiffness matrix** or discrete laplacian. Discretising in time with a **Runge-Kutta** method yields the numerical scheme:

$$\boldsymbol{U}^{n+1} = S\boldsymbol{U}^n + \Delta t \boldsymbol{F}^n$$

for some matrix S dependent on $\Delta t A$, hence on $\lambda = \Delta t/h^2$. For instance,

$$\boldsymbol{U}^{n+1} = (I - \Delta t A)\boldsymbol{U}^n + \Delta t \boldsymbol{F}^n_{\text{FE}}, \quad \text{(FE)}, \qquad \boldsymbol{U}^{n+1} = (I + \Delta t A)^{-1}\boldsymbol{U}^n + \Delta t \boldsymbol{F}^n_{\text{BE}}, \quad \text{(BE)}.$$

In this part of the talk we work in **bfloat16 half precision**, $u = 2^{-8} \approx 4 \times 10^{-3}$.

Use the delta form

How to best implement the Runge-Kutta scheme? Use the **delta form**!

$$\textbf{Standard form:} \quad \boldsymbol{U}^{n+1} = S\boldsymbol{U}^n + \Delta t \boldsymbol{F}^n.$$

$$\textbf{Delta form:} \quad \boldsymbol{U}^{n+1} = \boldsymbol{U}^n + \Delta t \left(-\tilde{S} A \boldsymbol{U}^n + \tilde{\boldsymbol{F}}^n \right) = \boldsymbol{U}^n + \Delta \boldsymbol{U}^n.$$

e.g. $S_{\mathsf{FE}} = (I - \Delta t A)$, $\tilde{S}_{FE} = 1$, and $S_{\mathsf{BE}} = \tilde{S}_{\mathsf{BE}} = (I + \Delta t A)^{-1}$.

Use the delta form

How to best implement the Runge-Kutta scheme? Use the **delta form**!

$$\textbf{Standard form:}\quad \boldsymbol{U}^{n+1} = S\boldsymbol{U}^n + \Delta t \boldsymbol{F}^n.$$

$$\textbf{Delta form:}\quad \boldsymbol{U}^{n+1} = \boldsymbol{U}^n + \Delta t \left(-\tilde{S}A\boldsymbol{U}^n + \tilde{\boldsymbol{F}}^n\right) = \boldsymbol{U}^n + \Delta \boldsymbol{U}^n.$$

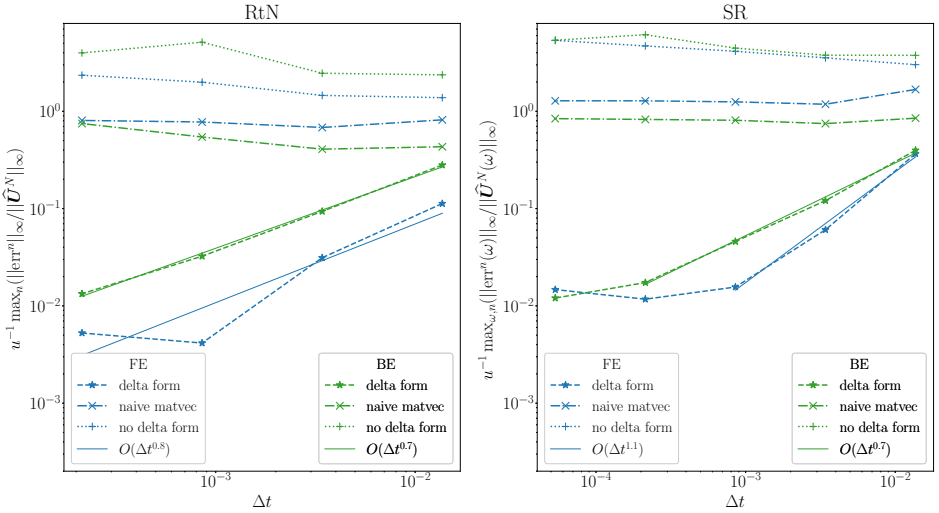
e.g. $S_{\mathsf{FE}} = (I - \Delta t A)$, $\tilde{S}_{FE} = 1$, and $S_{\mathsf{BE}} = \tilde{S}_{\mathsf{BE}} = (I + \Delta t A)^{-1}$.

- Errors in the computation of $S\boldsymbol{U}^n$ are of order u (machine precision).
- Errors in the computation of $\Delta \boldsymbol{U}^n$ are of order $\Delta t^p u$, $p > 0$.

We prove that:

- The delta form produces much smaller rounding errors at each time step.
- Most of the rounding errors in the delta form are introduced into the final addition.

Worst-case local rounding errors in 2D



Note: from now on we use the delta form.

RtN vs SR

Why is RtN in low precision bad for the heat equation?

a) Stagnation:

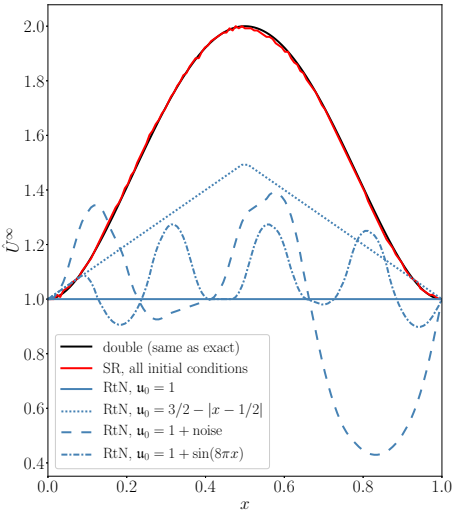
- RtN always stagnates for sufficiently small Δt (recall $\Delta \boldsymbol{U}^n = O(u\Delta t^p)$).
- The RtN solution is initial condition, discretization and precision dependent.

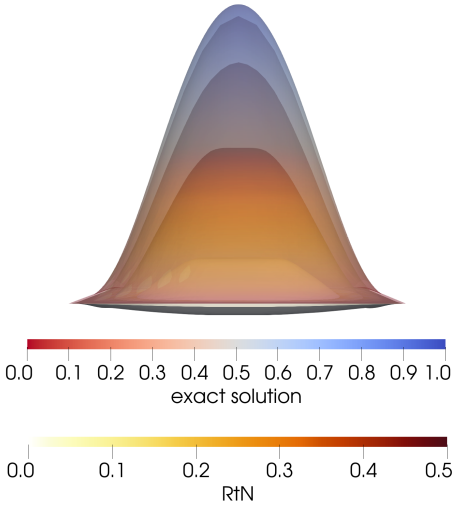
b) Global error:

- RtN rounding errors are strongly correlated and "amplify each other".
- RtN global errors grow like $O(u\Delta t^{-1})$ until stagnation.

SR fixes all these issues!

a) Stagnation (left 1D, right 2D)





RtN computations are discretization and initial condition dependent. SR works!

b) Global rounding errors [C. and Giles 2020]

Let $\varepsilon^n \in \mathbb{R}^K$ be the vector containing all rounding errors introduced at time step n.
Define the global rounding error $\boldsymbol{E}^n = \hat{\boldsymbol{U}}^n - \boldsymbol{U}^n$. It can be shown that

$$\boldsymbol{E}^{n+1} = S\boldsymbol{E}^n + \varepsilon^n.$$

Traditional results for ODEs [Henrici 1962-1963, Arató 1983]: ε^n is $O(\Delta t^2)$.

We can distinguish two cases:

RtN: we can only assume the worst-case scenario, $|\varepsilon_i^n| = O(u)$ for all n, i.

SR: the ε_i^n are zero-mean, independent in space and mean-independent in time.

b) Global rounding errors [C. and Giles 2020]

Let $\varepsilon^n \in \mathbb{R}^K$ be the vector containing all rounding errors introduced at time step n. Define the global rounding error $\boldsymbol{E}^n = \hat{\boldsymbol{U}}^n - \boldsymbol{U}^n$. It can be shown that

$$\boldsymbol{E}^{n+1} = S\boldsymbol{E}^n + \varepsilon^n.$$

Traditional results for ODEs [Henrici 1962-1963, Arató 1983]: ε^n is $O(\Delta t^2)$.

We can distinguish two cases:

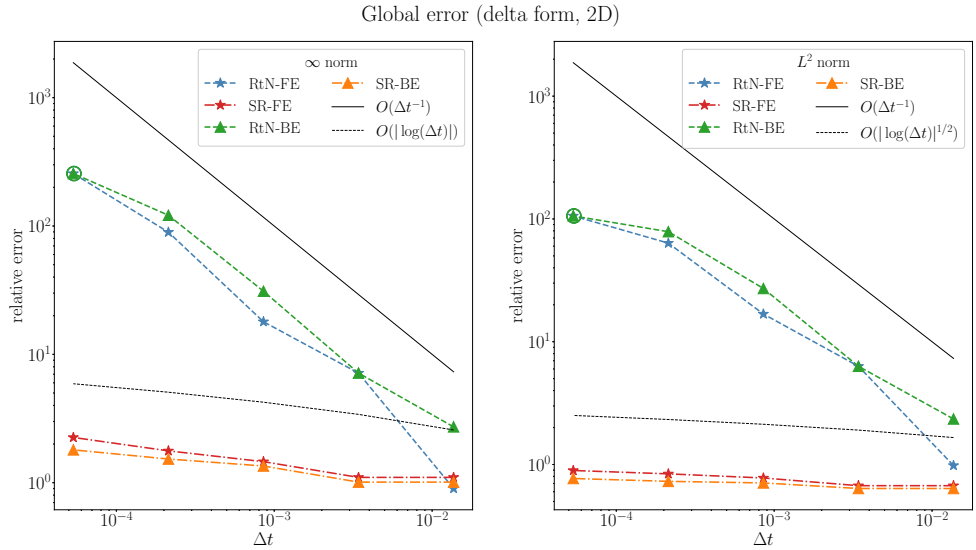
RtN: we can only assume the worst-case scenario, $|\varepsilon_i^n| = O(u)$ for all n, i.

SR: the ε_i^n are zero-mean, independent in space and mean-independent in time.

Mode	Norm	1D	2D	3D
RtN	L^2, ∞	$O(u\Delta t^{-1})$	$O(u\Delta t^{-1})$	$O(u\Delta t^{-1})$
SR	$\mathbb{E}[\|\|\cdot\|\|_\infty^2]^{1/2}$	$O(u\Delta t^{-1/4}\ell(\Delta t)^{1/2})$	$O(u\ell(\Delta t))$	$O(u\ell(\Delta t)^{1/2})$
SR	$\mathbb{E}[\|\|\cdot\|\|_{L^2}^2]^{1/2}$	$O(u\Delta t^{-1/4})$	$O(u\ell(\Delta t)^{1/2})$	$O(u)$

Asymptotic global rounding error blow-up rates; $\ell(\Delta t) = |\log(\lambda^{-1}\Delta t)|$.

b) Global rounding errors (here at steady-state)



Note: relative error = error $\times$ $(u||\boldsymbol{U}^N||)^{-1}$

Overview

1. Introduction and background

2. Solving parabolic PDEs in half precision (joint with M. B. Giles)

3. Mixed-precision explicit stabilised Runge-Kutta methods (with G. Rosilho De Souza)

4. Conclusions

Framework and objective

We consider mixed-precision explicit RK schemes for the solution of ODEs in the form

$$\boldsymbol{y}'(t) = \boldsymbol{f}(t, \boldsymbol{y}(t)), \quad \boldsymbol{y}(0) = \boldsymbol{y}_0,$$

where $\boldsymbol{f}(t, \boldsymbol{y})$ is of sufficiently smooth, and from now on set $\boldsymbol{f} = \boldsymbol{f}(\boldsymbol{y}(t))$ for simplicity. In our experiments: the ODE is a result of a method-of-lines discretisation of a PDE.

Objective

Evaluate $\boldsymbol{f}$ in low-precision as much as possible without affecting accuracy or stability.

Framework and objective

We consider mixed-precision explicit RK schemes for the solution of ODEs in the form

$$\boldsymbol{y}'(t) = \boldsymbol{f}(t, \boldsymbol{y}(t)), \quad \boldsymbol{y}(0) = \boldsymbol{y}_0,$$

where $\boldsymbol{f}(t, \boldsymbol{y})$ is of sufficiently smooth, and from now on set $\boldsymbol{f} = \boldsymbol{f}(\boldsymbol{y}(t))$ for simplicity. In our experiments: the ODE is a result of a method-of-lines discretisation of a PDE.

Objective

Evaluate $\boldsymbol{f}$ in low-precision as much as possible without affecting accuracy or stability.

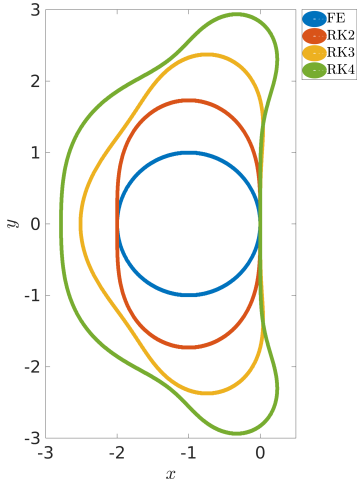
Why do we focus on explicit methods?

- The development of mixed-/reduced-precision linear/nonlinear iterative solvers is a very active field of research. Lots of exciting new work: [Abdelfattah et al. 2020].
- Avoiding nonlinear/linear solves is in general a big advantage. We use stabilised RK methods to minimise timestep restrictions.

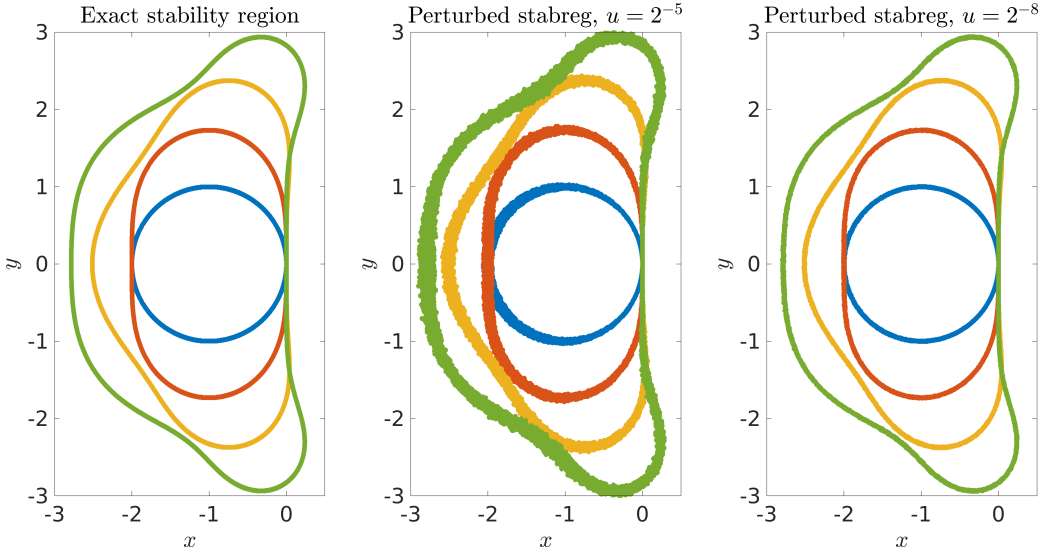
Note: in this part of the talk we only use RtN.

Absolute stability

Dahlquist's test problem: $y' = \lambda y$, $y(0) = 1$.
s-stage RK method $y^n = R_s(z)^n$, where $z = \Delta t \lambda = x + iy$. Stable if $|R_s(z)| < 1$.



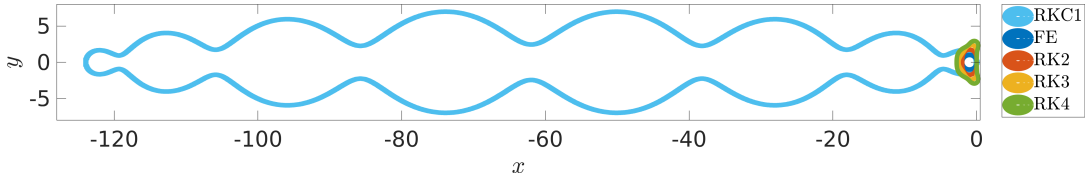
Linear stability for RK methods (in practice)



Explicit stabilized Runge-Kutta methods[2]

Idea: pick the poly $R_s(z)$ so as to maximise the stability region. For parabolic problems: use orthogonal polys, e.g. Runge-Kutta-Chebyshev (RKC). $\rightsquigarrow O(s^2)$ region.

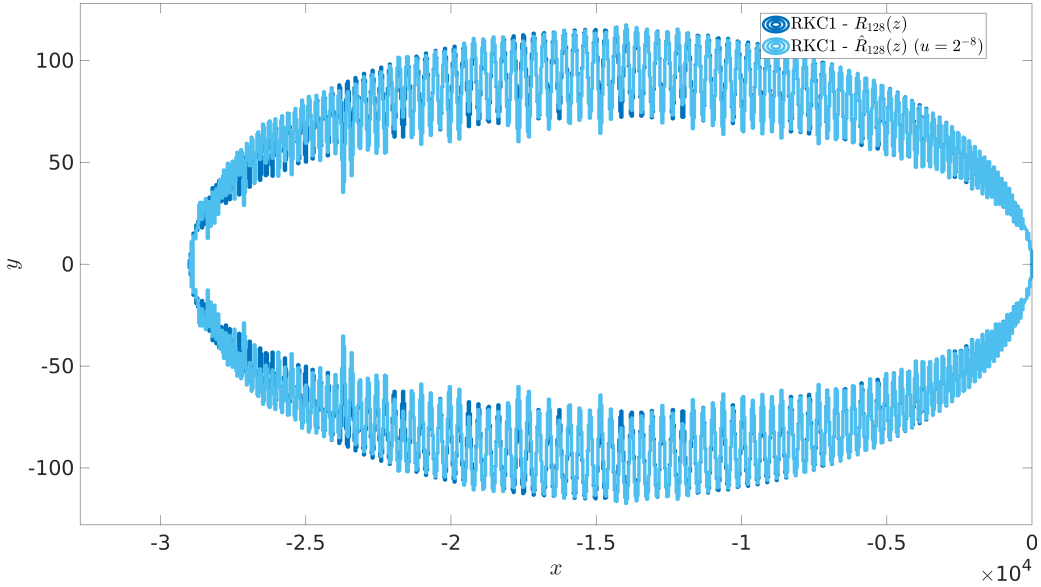
ESRK methods are of low-order ($p \leq 4$), but they use a lot of stages to maximise stability (i.e. not for accuracy purposes) $\rightarrow$ can do most of these in low precision!



Absolute stability region of RKC1 with $s = 8$ vs those of other explicit methods.

[2]Refs: [van der Houwen and Sommeijer 1980], many papers by Abdulle and collaborators.

Linear stability for RKC (in practice, $s = 128$, $u = 2^{-8}$)



Linear problems, i.e. $\boldsymbol{f}(\boldsymbol{y}) = A\boldsymbol{y}$

Consider the exact solution at $t = \Delta t$ and its corresponding p-th order RK approximation:

$$\boldsymbol{y}(\Delta t) = \exp(\Delta t A)\boldsymbol{y}_0 = \sum_{j=0}^{\infty} \frac{(\Delta t A)^j}{j!}\boldsymbol{y}_0,$$

$$\boldsymbol{y}_1 = \sum_{j=0}^{p} \frac{(\Delta t A)^j}{j!}\boldsymbol{y}_0 + O(\Delta t^{p+1}).$$

Giving a local error of $\tau = \Delta t^{-1}||\boldsymbol{y}(\Delta t) - \boldsymbol{y}_1|| = O(\Delta t^p)$.

Linear problems, i.e. $\boldsymbol{f}(\boldsymbol{y}) = A\boldsymbol{y}$

Consider the exact solution at $t = \Delta t$ and its corresponding p-th order RK approximation:

$$\boldsymbol{y}(\Delta t) = \exp(\Delta t A)\boldsymbol{y}_0 = \sum_{j=0}^{\infty} \frac{(\Delta t A)^j}{j!}\boldsymbol{y}_0,$$

$$\boldsymbol{y}_1 = \sum_{j=0}^{p} \frac{(\Delta t A)^j}{j!}\boldsymbol{y}_0 + O(\Delta t^{p+1}).$$

Giving a local error of $\tau = \Delta t^{-1}||\boldsymbol{y}(\Delta t) - \boldsymbol{y}_1|| = O(\Delta t^p)$.

Evaluating the scheme in finite precision yields:

$$\hat{\boldsymbol{y}}_1 = \varepsilon + \boldsymbol{y}_0 + \sum_{j=1}^{p} \frac{\Delta t^j}{j!} \left(\prod_{k=1}^{j} (A + \Delta A_k) \right) \boldsymbol{y}_0 + O(\Delta t^{p+1}).$$

Linear problems

$$\tau = \Delta^{-1}||\hat{\boldsymbol{y}}_1 - \boldsymbol{y}_1|| = \Delta t^{-1} \left|\left| \varepsilon + \sum_{j=1}^{p} \frac{\Delta t^j}{j!} \left(\prod_{k=1}^{j} (A + \Delta A_k) - A^j \right) \boldsymbol{y}_0 \right|\right| + O(\Delta t^p).$$

Linear problems

$$\tau = \Delta^{-1}||\hat{\boldsymbol{y}}_1 - \boldsymbol{y}_1|| = \Delta t^{-1} \left|\left| \varepsilon + \sum_{j=1}^{p} \frac{\Delta t^j}{j!} \left(\prod_{k=1}^{j} (A + \Delta A_k) - A^j \right) \boldsymbol{y}_0 \right|\right| + O(\Delta t^p).$$

Let us consider the following scenarios:

1. We have $\varepsilon = O(u)$ and we get $\tau = O(u\Delta t^{-1} + \Delta t^p)$. Rapid error growth!

Linear problems

$$\tau = \Delta^{-1}||\hat{\boldsymbol{y}}_1 - \boldsymbol{y}_1|| = \Delta t^{-1} \left|\left| \varepsilon + \sum_{j=1}^{p} \frac{\Delta t^j}{j!} \left(\prod_{k=1}^{j} (A + \Delta A_k) - A^j \right) \boldsymbol{y}_0 \right|\right| + O(\Delta t^p).$$

Let us consider the following scenarios:

1. We have $\varepsilon = O(u)$ and we get $\tau = O(u\Delta t^{-1} + \Delta t^p)$. Rapid error growth!
2. Exact vector operations: $\varepsilon = 0$ so $\tau = O(u + \Delta t^p)$. $O(u)$ limiting accuracy and loss of convergence.

Linear problems

$$\tau = \Delta^{-1}||\hat{\boldsymbol{y}}_1 - \boldsymbol{y}_1|| = \Delta t^{-1} \left|\left| \varepsilon + \sum_{j=1}^{p} \frac{\Delta t^j}{j!} \left(\prod_{k=1}^{j} (A + \Delta A_k) - A^j \right) \boldsymbol{y}_0 \right|\right| + O(\Delta t^p).$$

Let us consider the following scenarios:

1. We have $\varepsilon = O(u)$ and we get $\tau = O(u\Delta t^{-1} + \Delta t^p)$. Rapid error growth!
2. Exact vector operations: $\varepsilon = 0$ so $\tau = O(u + \Delta t^p)$. $O(u)$ limiting accuracy and loss of convergence.
3. First $q \geq 1$ matvecs exact. Now $\varepsilon = 0$ and $\Delta A_k = 0$ for $k = 1, \ldots, q$, so $\tau = O(u\Delta t^q + \Delta t^p)$. Recover q-th order convergence!

Order-preserving mixed-precision RK methods

From now on we set u to be the unit roundoff of the low-precision format.

Order-preserving mixed-precision RK methods

From now on we set u to be the unit roundoff of the low-precision format.

Assumption

Operations performed in high-precision are exact.

Definition (Order-preserving mixed-precision RK method)

A p-th order mixed-precision RK method is q-order-preserving ($q \in \{1, \dots, p\}$), if it converges with order q under the above assumption.

Order-preserving mixed-precision RK methods

From now on we set u to be the unit roundoff of the low-precision format.

Assumption

Operations performed in high-precision are exact.

Definition (Order-preserving mixed-precision RK method)

A p-th order mixed-precision RK method is q-order-preserving ($q \in \{1, \dots, p\}$), if it converges with order q under the above assumption.

Existing methods: Standard mixed-precision RK schemes perform all function evaluations in low-precision and they are therefore not order-preserving.

Our objective: Use q function evaluations in high precision to obtain a q-order-preserving mixed-precision RK method.

Result: We can construct q-order preserving RK methods for any q for linear problems, and for $q = 1, 2$ for nonlinear problems. Today we focus on RKC methods.

Mixed-precision RKC methods

One step of an s-stage RKC scheme in exact arithmetic is given by:

$$\begin{cases} \boldsymbol{d}_0 = \boldsymbol{0}, \quad \boldsymbol{d}_1 = \mu_1 \Delta t \boldsymbol{f}(\boldsymbol{y}^n), \\ \boldsymbol{d}_j = \nu_j \boldsymbol{d}_{j-1} + \kappa_j \boldsymbol{d}_{j-2} + \mu_j \Delta t \boldsymbol{f}(\boldsymbol{y}^n + \boldsymbol{d}_{j-1}) + \gamma_j \Delta t \boldsymbol{f}(\boldsymbol{y}^n), \quad j = 2, \ldots, s, \\ \boldsymbol{y}^{n+1} = \boldsymbol{y}^n + \boldsymbol{d}_s. \end{cases}$$

For a q-order preserving method we need to make sure all rounding errors are $O(\Delta t^{q+1})$.

Mixed-precision RKC methods

One step of a tentative mixed-precision scheme is given by:

$$\begin{cases} \hat{\boldsymbol{d}}_0 = \boldsymbol{0}, \quad \hat{\boldsymbol{d}}_1 = \mu_1 \Delta t \boldsymbol{f}(\hat{\boldsymbol{y}}^n), \\ \hat{\boldsymbol{d}}_j = \nu_j \hat{\boldsymbol{d}}_{j-1} + \kappa_j \hat{\boldsymbol{d}}_{j-2} + \mu_j \Delta t \hat{\boldsymbol{f}}(\hat{\boldsymbol{y}}^n + \hat{\boldsymbol{d}}_{j-1}) + \gamma_j \Delta t \boldsymbol{f}(\hat{\boldsymbol{y}}^n), \quad j = 2, \ldots, s, \\ \hat{\boldsymbol{y}}^{n+1} = \hat{\boldsymbol{y}}^n + \hat{\boldsymbol{d}}_s. \end{cases}$$

For a q-order preserving method we need to make sure all rounding errors are $O(\Delta t^{q+1})$.

The red term leads to an $O(u\Delta t)$ error! $\Rightarrow$ Must rewrite.

Mixed-precision RKC methods

We can rewrite:

$$\begin{cases} \hat{\boldsymbol{d}}_0 = \boldsymbol{0}, \quad \hat{\boldsymbol{d}}_1 = \mu_1 \Delta t \boldsymbol{f}(\hat{\boldsymbol{y}}^n), \\ \hat{\boldsymbol{d}}_j = \nu_j \hat{\boldsymbol{d}}_{j-1} + \kappa_j \hat{\boldsymbol{d}}_{j-2} + \mu_j \Delta t \hat{\Delta} \boldsymbol{f}_{j-1} + (\mu_j + \gamma_j) \Delta t \boldsymbol{f}(\hat{\boldsymbol{y}}^n), \quad j = 2, \dots, s, \\ \hat{\boldsymbol{y}}^{n+1} = \hat{\boldsymbol{y}}^n + \hat{\boldsymbol{d}}_s. \end{cases}$$

For a q-order preserving method we need to make sure all rounding errors are $O(\Delta t^{q+1})$.

The above is now a q-order preserving method as long as

$$\hat{\Delta} \boldsymbol{f}_j = \left(\boldsymbol{f}(\hat{\boldsymbol{y}}^n + \hat{\boldsymbol{d}}_{j-1}) - \boldsymbol{f}(\hat{\boldsymbol{y}}^n) \right) + O(\Delta t^q) = \Delta \boldsymbol{f}_j + O(\Delta t^q), \quad \forall j.$$

Note: if $\hat{\Delta} \boldsymbol{f}_j = \Delta \boldsymbol{f}_j$ we recover the exact RKC scheme.

Computing the $\hat{\Delta}\boldsymbol{f}_j$ terms - RKC1

For RKC1 we want

$$\hat{\Delta}\boldsymbol{f}_j = \Delta\boldsymbol{f}_j + O(\Delta t) = \Big(\boldsymbol{f}(\hat{\boldsymbol{y}}^n + \hat{\boldsymbol{d}}_{j-1}) - \boldsymbol{f}(\hat{\boldsymbol{y}}^n)\Big) + O(\Delta t), \quad \forall j.$$

Computing the $\hat{\Delta}\boldsymbol{f}_j$ terms - RKC1

For RKC1 we want

$$\hat{\Delta}\boldsymbol{f}_j = \Delta\boldsymbol{f}_j + O(\Delta t) = \Big(\boldsymbol{f}(\hat{\boldsymbol{y}}^n + \hat{\boldsymbol{d}}_{j-1}) - \boldsymbol{f}(\hat{\boldsymbol{y}}^n)\Big) + O(\Delta t), \quad \forall j.$$

It is sufficient that $\hat{\Delta}\boldsymbol{f}_j = \boldsymbol{f}'_j(\hat{\boldsymbol{y}}^n)\hat{\boldsymbol{d}}_j + O(\Delta t)$ since $\Delta\boldsymbol{f}_j = \boldsymbol{f}'_j(\hat{\boldsymbol{y}}^n)\hat{\boldsymbol{d}}_j + O(\Delta t)$.

Computing the $\hat{\Delta}\boldsymbol{f}_j$ terms - RKC1

For RKC1 we want

$$\hat{\Delta}\boldsymbol{f}_j = \Delta\boldsymbol{f}_j + O(\Delta t) = \Big(\boldsymbol{f}(\hat{\boldsymbol{y}}^n + \hat{\boldsymbol{d}}_{j-1}) - \boldsymbol{f}(\hat{\boldsymbol{y}}^n)\Big) + O(\Delta t), \quad \forall j.$$

It is sufficient that $\hat{\Delta}\boldsymbol{f}_j = \boldsymbol{f}'_j(\hat{\boldsymbol{y}}^n)\hat{\boldsymbol{d}}_j + O(\Delta t)$ since $\Delta\boldsymbol{f}_j = \boldsymbol{f}'_j(\hat{\boldsymbol{y}}^n)\hat{\boldsymbol{d}}_j + O(\Delta t)$.

Lots of options available to approximate/evaluate $\boldsymbol{f}'(\hat{\boldsymbol{y}}^n)$ (can use low precision).

Computing the $\hat{\Delta}\boldsymbol{f}_j$ terms - RKC2

For RKC2 we want

$$\hat{\Delta}\boldsymbol{f}_j = \Delta\boldsymbol{f}_j + O(\Delta t^2) = \Big(\boldsymbol{f}(\hat{\boldsymbol{y}}^n + \hat{\boldsymbol{d}}_{j-1}) - \boldsymbol{f}(\hat{\boldsymbol{y}}^n)\Big) + O(\Delta t^2), \quad \forall j.$$

Computing the $\hat{\Delta}\boldsymbol{f}_j$ terms - RKC2

For RKC2 we want

$$\hat{\Delta}\boldsymbol{f}_j = \Delta\boldsymbol{f}_j + O(\Delta t^2) = \Big(\boldsymbol{f}(\hat{\boldsymbol{y}}^n + \hat{\boldsymbol{d}}_{j-1}) - \boldsymbol{f}(\hat{\boldsymbol{y}}^n)\Big) + O(\Delta t^2), \quad \forall j.$$

Let $\hat{\boldsymbol{v}}_j = \hat{\boldsymbol{d}}_j - c_j\Delta t\boldsymbol{f}(\boldsymbol{y}^n)$, which is $O(\Delta t^2)$. We split $\Delta\boldsymbol{f}_j = \Delta_1\boldsymbol{f}_j + \Delta_2\boldsymbol{f}_j$, where

$$\begin{aligned}
\Delta_1\boldsymbol{f}_j &= \boldsymbol{f}(\hat{\boldsymbol{y}}^n + c_{j-1}\Delta t\boldsymbol{f}(\hat{\boldsymbol{y}}^n) + \hat{\boldsymbol{v}}_{j-1}) - \boldsymbol{f}(\hat{\boldsymbol{y}}^n + c_{j-1}\Delta t\boldsymbol{f}(\hat{\boldsymbol{y}}^n)),\\
\Delta_2\boldsymbol{f}_j &= \boldsymbol{f}(\hat{\boldsymbol{y}}^n + c_{j-1}\Delta t\boldsymbol{f}(\hat{\boldsymbol{y}}^n)) - \boldsymbol{f}(\hat{\boldsymbol{y}}^n).
\end{aligned}$$

Computing the $\hat{\Delta}\boldsymbol{f}_j$ terms - RKC2

For RKC2 we want

$$\hat{\Delta}\boldsymbol{f}_j = \Delta\boldsymbol{f}_j + O(\Delta t^2) = \Big(\boldsymbol{f}(\hat{\boldsymbol{y}}^n + \hat{\boldsymbol{d}}_{j-1}) - \boldsymbol{f}(\hat{\boldsymbol{y}}^n)\Big) + O(\Delta t^2), \quad \forall j.$$

Let $\hat{\boldsymbol{v}}_j = \hat{\boldsymbol{d}}_j - c_j\Delta t\boldsymbol{f}(\boldsymbol{y}^n)$, which is $O(\Delta t^2)$. We split $\Delta\boldsymbol{f}_j = \Delta_1\boldsymbol{f}_j + \Delta_2\boldsymbol{f}_j$, where

$$\begin{aligned}\Delta_1\boldsymbol{f}_j &= \boldsymbol{f}(\hat{\boldsymbol{y}}^n + c_{j-1}\Delta t\boldsymbol{f}(\hat{\boldsymbol{y}}^n) + \hat{\boldsymbol{v}}_{j-1}) - \boldsymbol{f}(\hat{\boldsymbol{y}}^n + c_{j-1}\Delta t\boldsymbol{f}(\hat{\boldsymbol{y}}^n)),\\ \Delta_2\boldsymbol{f}_j &= \boldsymbol{f}(\hat{\boldsymbol{y}}^n + c_{j-1}\Delta t\boldsymbol{f}(\hat{\boldsymbol{y}}^n)) - \boldsymbol{f}(\hat{\boldsymbol{y}}^n).\end{aligned}$$

It is sufficient to approximate both terms with $O(\Delta t^2)$ error. Again we use derivatives:

$$\begin{aligned}\hat{\Delta}_1\boldsymbol{f}_j &= \hat{\boldsymbol{f}}'(\hat{\boldsymbol{y}}^n + c_j\Delta t\boldsymbol{f}(\hat{\boldsymbol{y}}^n))\hat{\boldsymbol{v}}_j = \Delta_1\boldsymbol{f}_j + O(\Delta t^2),\\ \hat{\Delta}_2\boldsymbol{f}_j &= c_j\Delta t\boldsymbol{f}'(\hat{\boldsymbol{y}}^n)\boldsymbol{f}(\hat{\boldsymbol{y}}^n) = \Delta_2\boldsymbol{f}_j + O(\Delta t^2),\end{aligned}$$

Linear stability

Theorem

Let $\boldsymbol{f}(\boldsymbol{y}) = A\boldsymbol{y}$ with A being a symmetric npd matrix. Our order-p schemes satisfy

$$\hat{\boldsymbol{y}}^{n+1} = R_s^p(\Delta t A)\hat{\boldsymbol{y}}^n + \boldsymbol{r}_s^p(\hat{\boldsymbol{y}}^n),$$

where $\boldsymbol{r}_s^p$ contains the rounding errors introduced at time step n,

Linear stability

Theorem

Let $\boldsymbol{f}(\boldsymbol{y}) = A\boldsymbol{y}$ with A being a symmetric npd matrix. Our order-p schemes satisfy

$$\hat{\boldsymbol{y}}^{n+1} = R_s^p(\Delta t A)\hat{\boldsymbol{y}}^n + \boldsymbol{r}_s^p(\hat{\boldsymbol{y}}^n),$$

where $\boldsymbol{r}_s^p$ contains the rounding errors introduced at time step n, and is bounded by

$$\|\boldsymbol{r}_s^p\|_2 \leq \Psi_p(\Delta t, A)\left((1 + C_p(s,\varepsilon)\Delta t u)^{s-1} - 1\right)\|\hat{\boldsymbol{y}}^n\|_2.$$

Linear stability

Theorem

Let $\boldsymbol{f}(\boldsymbol{y}) = A\boldsymbol{y}$ with A being a symmetric npd matrix. Our order-p schemes satisfy

$$\hat{\boldsymbol{y}}^{n+1} = R_s^p(\Delta t A)\hat{\boldsymbol{y}}^n + \boldsymbol{r}_s^p(\hat{\boldsymbol{y}}^n),$$

where $\boldsymbol{r}_s^p$ contains the rounding errors introduced at time step n, and is bounded by

$$\|\boldsymbol{r}_s^p\|_2 \leq \Psi_p(\Delta t, A)\left((1 + C_p(s,\varepsilon)\Delta t u)^{s-1} - 1\right)\|\hat{\boldsymbol{y}}^n\|_2.$$

$\Psi_p(\Delta t, A)$ is method-dependent and given by

$$\Psi_1(\Delta t, A) = \max_{k=1,\ldots,s-1} \|R_k^p(\Delta t A) - I\|_2 = O(\Delta t), \qquad \text{for RKC1},$$

$$\Psi_2(\Delta t, A) = \max_{k=1,\ldots,s-1} \|R_k^p(\Delta t A) - I - c_k \Delta t A\|_2 = O(\Delta t^2), \qquad \text{for RKC2}.$$

Linear stability

Theorem

Let $\boldsymbol{f}(\boldsymbol{y}) = A\boldsymbol{y}$ with A being a symmetric npd matrix. Our order-p schemes satisfy

$$\hat{\boldsymbol{y}}^{n+1} = R_s^p(\Delta t A)\hat{\boldsymbol{y}}^n + \boldsymbol{r}_s^p(\hat{\boldsymbol{y}}^n),$$

where $\boldsymbol{r}_s^p$ contains the rounding errors introduced at time step n, and is bounded by

$$\|\boldsymbol{r}_s^p\|_2 \leq \Psi_p(\Delta t, A)\left((1 + C_p(s,\varepsilon)\Delta t u)^{s-1} - 1\right)\|\hat{\boldsymbol{y}}^n\|_2.$$

$\Psi_p(\Delta t, A)$ is method-dependent and given by

$$\Psi_1(\Delta t, A) = \max_{k=1,\ldots,s-1} \|R_k^p(\Delta t A) - I\|_2 = O(\Delta t), \qquad \text{for RKC1},$$

$$\Psi_2(\Delta t, A) = \max_{k=1,\ldots,s-1} \|R_k^p(\Delta t A) - I - c_k \Delta t A\|_2 = O(\Delta t^2), \qquad \text{for RKC2}.$$

Note: For large Δt and s we have $\Psi_1(\Delta t A) \leq 2$, $\Psi_2(\Delta t A) \leq 2 + \Delta t\|A\|_2 \leq cs^2$.
$\Rightarrow$ Mixed-precision RKC2 is unstable for large Δt and s.

Stabilising RKC2

Recall: we computed $\hat{\Delta}\boldsymbol{f}_j = \hat{\Delta}_1\boldsymbol{f}_j + \hat{\Delta}_2\boldsymbol{f}_j$, where $\hat{\Delta}_1\boldsymbol{f}_j = \boldsymbol{f}'(\dots)\hat{\boldsymbol{v}}_j = O(\Delta t^2)$.

The culprit is the $\hat{\boldsymbol{v}}_j$ term: for small Δt this is small and ensures 2nd order convergence, but for large Δt it becomes huge and leads to instability!

Stabilising RKC2

Recall: we computed $\hat{\Delta}\boldsymbol{f}_j = \hat{\Delta}_1\boldsymbol{f}_j + \hat{\Delta}_2\boldsymbol{f}_j$, where $\hat{\Delta}_1\boldsymbol{f}_j = \boldsymbol{f}'(\dots)\hat{\boldsymbol{v}}_j = O(\Delta t^2)$.

The culprit is the $\hat{\boldsymbol{v}}_j$ term: for small Δt this is small and ensures 2nd order convergence, but for large Δt it becomes huge and leads to instability!

To fix this, consider the 1-order preserving evaluation of $\hat{\Delta}\boldsymbol{f}_j$ (same as for RKC1):

$$\tilde{\Delta}\boldsymbol{f}_j = \boldsymbol{f}'_j(\hat{\boldsymbol{y}}_j)\hat{\boldsymbol{d}}_j + O(\Delta t).$$

This leads to a stable scheme for large Δt, but is only first-order accurate for small Δt.

Stabilising RKC2

Recall: we computed $\hat{\Delta}\boldsymbol{f}_j = \hat{\Delta}_1\boldsymbol{f}_j + \hat{\Delta}_2\boldsymbol{f}_j$, where $\hat{\Delta}_1\boldsymbol{f}_j = \boldsymbol{f}'(\dots)\hat{\boldsymbol{v}}_j = O(\Delta t^2)$.

The culprit is the $\hat{\boldsymbol{v}}_j$ term: for small Δt this is small and ensures 2nd order convergence, but for large Δt it becomes huge and leads to instability!

To fix this, consider the 1-order preserving evaluation of $\hat{\Delta}\boldsymbol{f}_j$ (same as for RKC1):

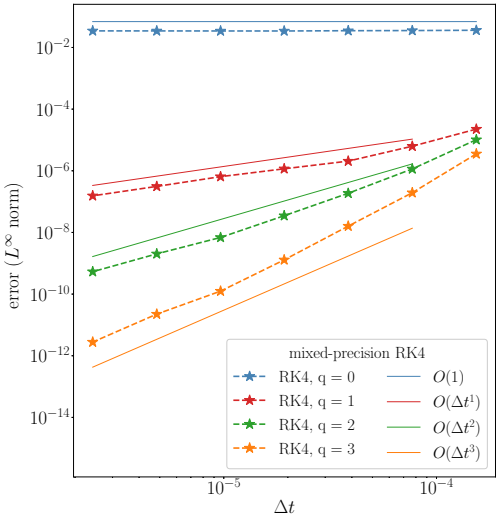
$$\tilde{\Delta}\boldsymbol{f}_j = \boldsymbol{f}'_j(\hat{\boldsymbol{y}}_j)\hat{\boldsymbol{d}}_j + O(\Delta t).$$

This leads to a stable scheme for large Δt, but is only first-order accurate for small Δt.

Solution: set $\hat{\Delta}\boldsymbol{f}_j = \begin{cases} \hat{\Delta}_1\boldsymbol{f}_j + \hat{\Delta}_2\boldsymbol{f}_j, & \text{if } \|\hat{\boldsymbol{v}}_j\|_2 \le \|\hat{\boldsymbol{d}}_j\|_2, \\ \tilde{\Delta}\boldsymbol{f}_j, & \text{if } \|\hat{\boldsymbol{v}}_j\|_2 > \|\hat{\boldsymbol{d}}_j\|_2. \end{cases}$

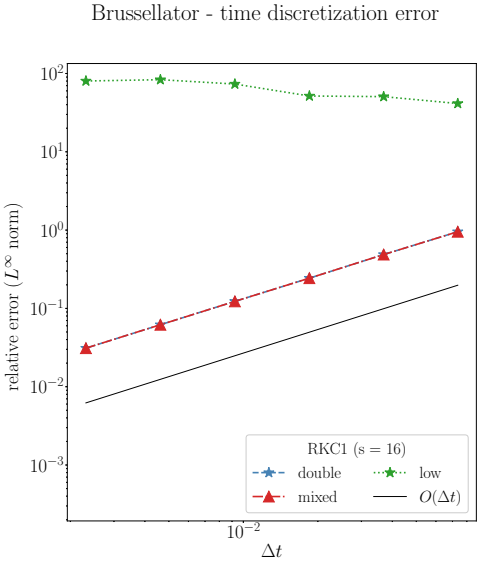
This leads to a 2nd-order *and* stable method. Can prove $\Psi_{2,\text{new}} = \min(\Psi_1, \Psi_2) \le 2$.

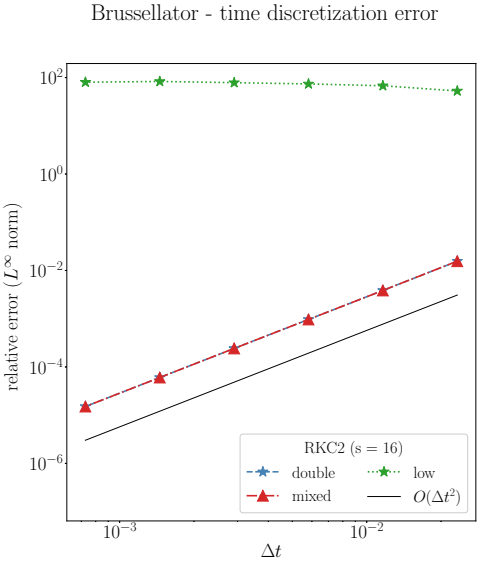
Numerical results - convergence (3D heat eqn, half precision)



The transition from order p to order q happens roughly when $\Delta t = O(||A||^{-1} u^{\frac{1}{p-q}})$

Numerical results - convergence (Brussellator, half precision)





Overview

1. Introduction and background

2. Solving parabolic PDEs in half precision (joint with M. B. Giles)

3. Mixed-precision explicit stabilised Runge-Kutta methods (with G. Rosilho De Souza)

4. Conclusions

Outlook

To sum up

- Reduced-/mixed-precision algorithms require a careful implementation, but can bring significant memory, cost, and energy savings.
- SR makes computations very resilient to stagnation and error accumulation. If used correctly it can make reduced-precision computations very robust.
- We can make ESRK methods as accurate as their high precision equivalent and as cheap as their fully low-precision counterpart.
- Our work extends to multirate ESRK and to RK methods in general for $q = 1, 2$.
- We used a custom-built C++ low-precision emulator for most results.

Outlook

To sum up

- Reduced-/mixed-precision algorithms require a careful implementation, but can bring significant memory, cost, and energy savings.
- SR makes computations very resilient to stagnation and error accumulation. If used correctly it can make reduced-precision computations very robust.
- We can make ESRK methods as accurate as their high precision equivalent and as cheap as their fully low-precision counterpart.
- Our work extends to multirate ESRK and to RK methods in general for $q = 1, 2$.
- We used a custom-built C++ low-precision emulator for most results.

Thank you for listening!

Thank you for listening! If you want to know more...

Papers and more info at: `https://croci.github.io`

References

[1] M. Croci and M. B. Giles. Effects of round-to-nearest and stochastic rounding in the numerical solution of the heat equation in low precision. *IMA Journal of Numerical Analysis (to appear)*, 2022. URL `https://arxiv.org/pdf/2010.16225`.

[2] M. Croci, M. Fasi, N. J. Higham, T. Mary, and M. Mikaitis. Stochastic rounding: implementation, error analysis and applications. *Royal Society Open Science*, 9:211631, 2022.

[3] M. Croci and G. R. de Souza. Mixed-precision explicit stabilized Runge-Kutta methods for single- and multi-scale differential equations, 2021. URL `https://arxiv.org/pdf/2109.12153`.

[4] M. Klöwer, S. Hatfield, M. Croci, P. D. Düben, and T. N. Palmer. Fluid simulations accelerated with 16 bits: Approaching 4x speedup on A64FX by squeezing ShallowWaters.jl into Float16. *Journal of Advances in Modeling Earth Systems*, 2021.

[5] A. Abdelfattah, H. Anzt, E. G. Boman, E. Carson, T. Cojean, J. Dongarra, A. Fox, M. Gates, N. J. Higham, X. S. Li, et al. A survey of numerical linear algebra methods utilizing mixed-precision arithmetic. *The International Journal of High Performance Computing Applications*, 35(4):344–369, 2021.

[6] N. J. Higham and T. Mary. Mixed precision algorithms in numerical linear algebra, 2021. URL `http://eprints.maths.manchester.ac.uk/2841/1/paper_eprint.pdf`.

[7] M. P. Connolly, N. J. Higham, and T. Mary. Stochastic rounding and its probabilistic backward error analysis. *SIAM Journal on Scientific Computing*, 43(1):566–585, 2021.

[8] J. G. Verwer, W. H. Hundsdorfer, and B. P. Sommeijer. Convergence properties of the Runge-Kutta-Chebyshev method. *Numerische Mathematik*, 57:157–178, 1990.

APPENDIX

Exploit exact subtraction

How to best implement the matrix-vector product $-A\boldsymbol{U}^n$?

$$\frac{\boldsymbol{U}_{i+1}^n - 2\boldsymbol{U}_i^n + \boldsymbol{U}_{i-1}^n}{h^2}, \qquad \frac{(\boldsymbol{U}_{i+1}^n - \boldsymbol{U}_i^n) - (\boldsymbol{U}_i^n - \boldsymbol{U}_{i-1}^n)}{h^2}.$$

Exploit exact subtraction

How to best implement the matrix-vector product $-A\boldsymbol{U}^n$?

$$\frac{\boldsymbol{U}_{i+1}^n - 2\boldsymbol{U}_i^n + \boldsymbol{U}_{i-1}^n}{h^2}, \qquad \frac{(\boldsymbol{U}_{i+1}^n - \boldsymbol{U}_i^n) - (\boldsymbol{U}_i^n - \boldsymbol{U}_{i-1}^n)}{h^2}.$$

Leads to $O(h^{-2})$ error! **Leads to near-exact matvecs.**

A similar trick works for FEM as well. Only requires small modification of CSR matvecs.

Exploit exact subtraction

How to best implement the matrix-vector product $-A\boldsymbol{U}^n$?

$$\frac{\boldsymbol{U}_{i+1}^n - 2\boldsymbol{U}_i^n + \boldsymbol{U}_{i-1}^n}{h^2}, \qquad \frac{(\boldsymbol{U}_{i+1}^n - \boldsymbol{U}_i^n) - (\boldsymbol{U}_i^n - \boldsymbol{U}_{i-1}^n)}{h^2}.$$

Leads to $O(h^{-2})$ error! **Leads to near-exact matvecs.**

A similar trick works for FEM as well. Only requires small modification of CSR matvecs.

Parts of a Theorem [C. and Giles 2020]

If $a, b \in \mathbb{R}$ are exactly represented in floating point arithmetic, and

$$|a - b| \leq \min(|a|, |b|)$$

then $(a - b)$ is computed exactly.

See also Section 2.5 in "Accuracy and Stability of Numerical Algorithms" by Nick Higham.

Computational savings

The mixed-precision scheme is cheaper by roughly a factor

$$\varrho = \frac{(s-q)(r-1)}{sr}, \quad \text{where} \quad r = \frac{\text{Cost of RHS evals in high}}{\text{Cost of RHS evals in low}}.$$

A scheme in double/half yields $r = 4^\ell$ for $O(N^\ell)$-cost RHS evaluations.

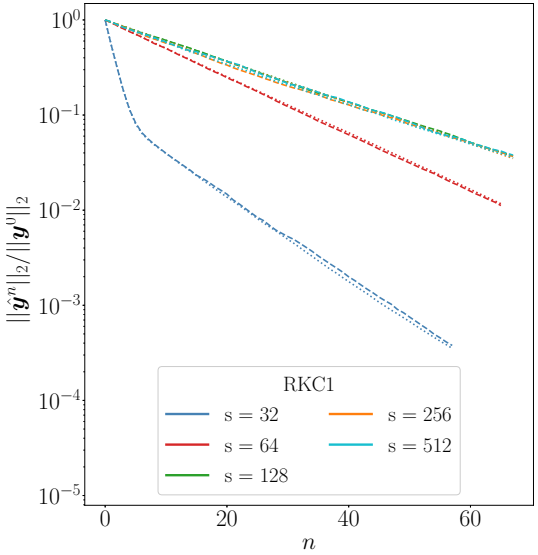
- For RK4 and $\ell = 1$ this leads to 56% $(q = 1)$ and 40% $(q = 2)$ savings.
- Stabilised methods have lots of stages and low order: can essentially take $s \to \infty$, giving $\varrho \to 1 - 1/r$. E.g. this leads to 75% savings if $\ell = 1$.

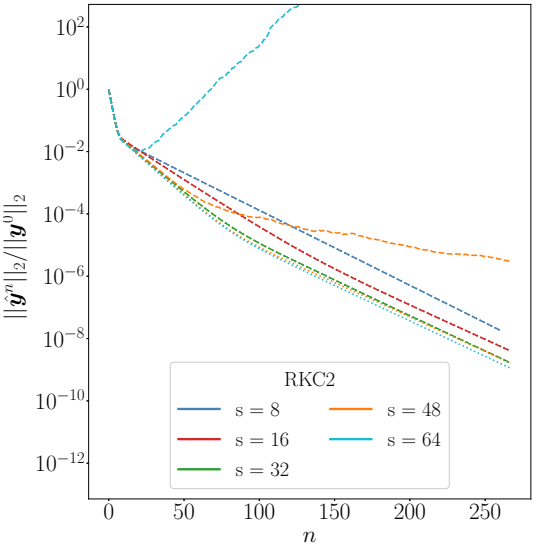
Note: We have ignored additional savings related to memory/caching effects.

Linear stability result - some comments

- The bounds account for rounding errors propagating from previous stages, an overlooked phenomenon in standard RKC theory [Verwer et al. 1990].
- No stability in the classical sense: local errors are amplified for large Δt.
- Rounding errors act on all frequencies and destroy any spectral relation between the iterates forbidding any stability analysis based on eigenvalues.
- We can only prove actual stability under stringent conditions on $\mathrm{cond}(A)$.
- The previous result is a very pessimistic worst-case bound that does not seem to affect computations in practice.

Numerical results - stability (2D nonlinear heat eqn, half precision)





Numerical results - error vs number of stages