May 29, 2015

Solving Differential Matrix Equations using Parareal

Martin Köhler

joint work with

Jens Saak and Norman Lang

PinT 2015
4th Workshop on Parallel – in – Time Integration

TUC TECHNISCHE UNIVERSITÄT CHEMNITZ

MAX PLANCK INSTITUTE FOR DYNAMICS OF COMPLEX TECHNICAL SYSTEMS MAGDEBURG

Outline

Motivation

Differential Riccati Equations

Consider the linear quadratic optimal control problem

$$\min_u \mathcal{J}(y,u) = \frac{1}{2}\left(\int_0^{t_f} y^T y + u^T u \,\mathrm{d}t \; + y_{t_f}^T Q y_{t_f}\right),$$

$$\text{subject to} \quad E\dot{x}(t) = Ax(t) + Bu(t),$$
$$y(t) = Cx(t)$$

where A, E, B, and C may depend on t as well.

with the states $x(t) \in \mathbb{R}^n$, inputs $u(t) \in \mathbb{R}^m$, and output $y(t) \in \mathbb{R}^q$.

Motivation

Differential Riccati Equations

Consider the linear quadratic optimal control problem

$$\min_{u} \mathcal{J}(y,u) = \frac{1}{2}\left(\int_0^{t_f} y^T y + u^T u \,\mathrm{d}t \; + y_{t_f}^T Q y_{t_f}\right),$$

$$\text{subject to} \quad E\dot{x}(t) = Ax(t) + Bu(t),$$
$$y(t) = Cx(t)$$

where A, E, B, and C may depend on t as well.

Feedback law

e.g. [Locatelli '01]

$$u(t) = -B^T X(t) E x(t),$$

Motivation

Differential Riccati Equations

Consider the linear quadratic optimal control problem

$$\min_u \mathcal{J}(y,u) = \frac{1}{2}\left(\int_0^{t_f} y^T y + u^T u \,\mathrm{d}t \; + y_{t_f}^T Q y_{t_f}\right),$$

$$\text{subject to} \quad E\dot{x}(t) = Ax(t) + Bu(t),$$
$$y(t) = Cx(t)$$

where A, E, B, and C may depend on t as well.

Feedback law

e.g. [Locatelli '01]

$$u(t) = -B^T X(t) E x(t),$$

where $X(t)$ is the solution of the Differential Riccati Equation (DRE)

$$E^T \dot{X}(t) E = C^T C + A^T X(t) E + E^T X(t) A - E^T X(t) B B^T X(t) E := \mathcal{R}(X(t)),$$
$$X(t = t_f) := Q.$$

Motivation

Differential Lyapunov Equation

Simplification of the DRE

By setting $B = 0$ in the DRE we get the Differential Lyapunov Equation (DLE):

$$E^T \dot{X}(t) E = C^T C + A^T X(t) E + E^T X(t) A,$$
$$X(t = t_f) := X_f.$$

Motivation

Differential Lyapunov Equation

Simplification of the DRE

By setting $B = 0$ in the DRE we get the Differential Lyapunov Equation (DLE):

$$E^T \dot{X}(t) E = C^T C + A^T X(t) E + E^T X(t) A,$$
$$X(t = t_f) := X_f.$$

Application in Model Order Reduction:

$\rightarrow$ used for *Linear Time-Variant (LTV)* Balanced Truncation.

Motivation

Time integration methods

- The DLE is a matrix-valued ordinary differential equation.

Motivation

Time integration methods

- The DLE is a matrix-valued ordinary differential equation.
- The DRE is a non-linear, matrix-valued, and highly stiff ordinary differential equation.

Motivation

Time integration methods

- The DLE is a matrix-valued ordinary differential equation.
- The DRE is a non-linear, matrix-valued, and highly stiff ordinary differential equation.

Implicit time integrators [Mena '07, Benner/Mena '12]

- Backward differentiation formula (BDF)
- Linear implicit Runge-Kutta (Rosenbrock) methods
- Midpoint and Trapezoidal rule

Motivation

Time integration methods

- The DLE is a matrix-valued ordinary differential equation.
- The DRE is a non-linear, matrix-valued, and highly stiff ordinary differential equation.

Implicit time integrators [Mena '07, Benner/Mena '12]

- Backward differentiation formula (BDF)
- Linear implicit Runge-Kutta (Rosenbrock) methods
- Midpoint and Trapezoidal rule

Numerical issues

- Methods are fairly time and storage consuming for large-scale problems.
- High accuracy requires small time steps or high order methods.
- At every time step a number of algebraic matrix equations needs to be solved.

Motivation

Time integration methods

- The DLE is a matrix-valued ordinary differential equation.
- The DRE is a non-linear, matrix-valued, and highly stiff ordinary differential equation.

Implicit time integrators [Mena '07, Benner/Mena '12]

- Backward differentiation formula (BDF)
- Linear implicit Runge-Kutta (Rosenbrock) methods
- Midpoint and Trapezoidal rule

Good reasons to try Parareal!

Numerical issues

- Methods are fairly time and storage consuming for large-scale problems.
- High accuracy requires small time steps or high order methods.
- At every time step a number of algebraic matrix equations needs to be solved.

Rosenbrock Methods

Restriction by Parareal:

Only single-step integrators are well suited.

Rosenbrock Methods

Restriction by Parareal:

Only single-step integrators are well suited.

General Rosenbrock Scheme

The s-stage Rosenbrock method applied to a matrix differential equation of the form $\dot{X} = F(X)$ is given as

$$X_{k+1} = X_k + \tau_k \sum_{\ell=1}^{s} b_\ell K_\ell^{(k)},$$

$$K_i^{(k)} = F(X_k + \tau_k \sum_{\ell=1}^{i-1} \alpha_{i,\ell} K_\ell^{(k)}) + \tau_k \mathcal{J}_k \sum_{\ell=1}^{i} \gamma_{i,\ell} K_\ell^{(k)}, \quad \forall i = 1, \ldots, s.$$

- s : order of the method
- τ_k : time step
- $\mathcal{J}_k$: Fréchet derivative of F at X_k
- $\alpha_{i,\ell}, \gamma_{i,\ell}, \mu_\ell$: determining coefficients

Rosenbrock Methods

[Mena '07, Benner/Mena '12]

We only consider the DRE, where $E^T \dot{X} E = \mathcal{R}(X)$:

1^{st} order Rosenbrock scheme (Ros1)

$$X_{k+1} = X_k + \tau_k K_1^{(k)}$$

Rosenbrock Methods

[Mena '07, Benner/Mena '12]

We only consider the DRE, where $E^T \dot{X} E = \mathcal{R}(X)$:

1^{st} order Rosenbrock scheme (Ros1)

$$X_{k+1} = X_k + \tau_k K_1^{(k)}$$

$$E^T K_1^{(k)} E - \tau_k \mathcal{R}'|_{X_k}(K_1^{(k)}) = \mathcal{R}(X)$$

Rosenbrock Methods

[Mena '07, Benner/Mena '12]

We only consider the DRE, where $E^T \dot{X} E = \mathcal{R}(X)$:

1^{st} order Rosenbrock scheme (Ros1)

$$X_{k+1} = X_k + \tau_k K_1^{(k)}$$

$$E^T K_1^{(k)} E - \tau_k (A - BB^T X_k E)^T K_1^{(k)} E - \tau_k E^T K_1^{(k)} (A - BB^T X_k E) = \mathcal{R}(X)$$

Rosenbrock Methods

[Mena '07, Benner/Mena '12]

We only consider the DRE, where $E^T \dot{X} E = \mathcal{R}(X)$:

1^{st} order Rosenbrock scheme (Ros1)

$$X_{k+1} = X_k + \tau_k K_1^{(k)}$$

$$(\tau_k(A - BB^T X_k E) - \frac{1}{2}E)^T K_1^{(k)} E + E^T K_1^{(k)} (\tau_k(A - BB^T X_k E) - \frac{1}{2}E) = -\mathcal{R}(X)$$

Rosenbrock Methods

[Mena '07, Benner/Mena '12]

We only consider the DRE, where $E^T \dot{X} E = \mathcal{R}(X)$:

1^{st} order Rosenbrock scheme (Ros1)

$$X_{k+1} = X_k + \tau_k K_1^{(k)}$$

$$\tilde{A}^T K_1^{(k)} E + E^T K_1^{(k)} \tilde{A} = -\mathcal{R}(X_k)$$

$$\tilde{A} := \tau_k (A - BB^T X_k E) - \frac{1}{2} E$$

Solve one Algebraic Lyapunov Equation (ALE) inside the 1-stage Rosenbrock method.

Rosenbrock Methods

[Mena '07, Benner/Mena '12]

We only consider the DRE, where $E^T \dot{X} E = \mathcal{R}(X)$:

1^{st} order Rosenbrock scheme (Ros1)

$$X_{k+1} = X_k + \tau_k K_1^{(k)}$$

$$\tilde{A}^T K_1^{(k)} E + E^T K_1^{(k)} \tilde{A} = -\mathcal{R}(X_k)$$

2^{nd} order Rosenbrock scheme (Ros2)

[Dekker, Verwer '84]

$$X_{k+1} = X_k + \frac{3}{2}\tau_k K_1^{(k)} + \frac{1}{2}\tau_k K_2^{(k)}$$

$$\tilde{A}^T K_1^{(k)} E + E^T K_1^{(k)} \tilde{A} = -\mathcal{R}(X_k)$$

$$\tilde{A}^T K_2^{(k)} E + E^T K_2^{(k)} \tilde{A} = -\mathcal{R}(X_k + \tau_k K_1^{(k)}) + 2E^T K_1^{(k)} E$$

$$\tilde{A} := \gamma\tau_k (A - BB^T X_k E) - \frac{1}{2} E$$

Rosenbrock Methods

[Mena '07, Benner/Mena '12]

We only consider the DRE, where $E^T \dot{X} E = \mathcal{R}(X)$:

1^{st} order Rosenbrock scheme (Ros1)

$$X_{k+1} = X_k + \tau_k K_1^{(k)}$$

$$\tilde{A}^T K_1^{(k)} E + E^T K_1^{(k)} \tilde{A} = -\mathcal{R}(X_k)$$

2^{nd} order Rosenbrock scheme (Ros2)

[Dekker, Verwer '84]

$$X_{k+1} = X_k + \frac{3}{2}\tau_k K_1^{(k)} + \frac{1}{2}\tau_k K_2^{(k)}$$

$$\tilde{A}^T K_1^{(k)} E + E^T K_1^{(k)} \tilde{A} = -\mathcal{R}(X_k)$$

$$\tilde{A}^T K_2^{(k)} E + E^T K_2^{(k)} \tilde{A} = -\mathcal{R}(X_k + \tau_k K_1^{(k)}) + 2E^T K_1^{(k)} E$$

Solve two ALEs inside the 2-stage Rosenbrock method.

Rosenbrock Methods

Higher Order Schemes

3^{rd} order scheme (Ros3) [Ros3P: Lang, Verwer '01]

$$X_{k+1} = X_k + \tau_k \sum_{j=1}^{s} \mu_j K_j,$$

$$\begin{aligned}
\tilde{A}_k K_1 E^T + E K_1 \tilde{A}_k^T &= -\mathcal{R}(X_k), \\
\tilde{A}_k K_2 E^T + E K_2 \tilde{A}_k^T &= -\mathcal{R}(X_k + \tau_k a_{21} K_1) - c_{21} E K_1 E^T, \\
\tilde{A}_k K_3 E^T + E K_3 \tilde{A}_k^T &= -\mathcal{R}(X_k + \tau_k a_{31} K_1) - c_{31} E K_1 E^T - c_{32} E K_2 E^T.
\end{aligned}$$

with $\tilde{A}_k := \tau_k (A - BB^T X_k E) - \frac{1}{2\gamma} E$.

Rosenbrock Methods

Higher Order Schemes

3^{rd} order scheme (Ros3) [Ros3P: Lang, Verwer '01]

$$X_{k+1} = X_k + \tau_k \sum_{j=1}^{s} \mu_j K_j,$$

$$\tilde{A}_k K_1 E^T + E K_1 \tilde{A}_k^T = -\mathcal{R}(X_k),$$
$$\tilde{A}_k K_2 E^T + E K_2 \tilde{A}_k^T = -\mathcal{R}(X_k + \tau_k a_{21} K_1) - c_{21} E K_1 E^T,$$
$$\tilde{A}_k K_3 E^T + E K_3 \tilde{A}_k^T = -\mathcal{R}(X_k + \tau_k a_{31} K_1) - c_{31} E K_1 E^T - c_{32} E K_2 E^T.$$

Determining Coefficients

$\gamma = 7.886751345948129e\text{-}1$	
$a_{21} = 1.267949192431123$ $a_{31} = 1.267949192431123$ $a_{32} = 0$	$\alpha_1 = 0$ $\alpha_2 = 1$ $\alpha_3 = 1$
$c_{21} = -1.607695154586736$ $c_{31} = -3.464101615137755$ $c_{32} = -1.732050807568877$	$\mu_1 = 2$ $\mu_2 = 5.773502691896258e-1$ $\mu_3 = 4.226497308103742e-1$

Rosenbrock Methods

Higher Order Schemes

4^{th} order scheme (Ros4) [Shampine '82]

$$
\begin{aligned}
X_{k+1} =& X_k + \tau_k \sum_{j=1}^{4} b_j K_j, \\
\tilde{A}_k^T K_1 E + E^T K_1 \tilde{A}_k =& -\mathcal{R}(X_k), \\
\tilde{A}_k^T K_2 E + E^T K_2 \tilde{A}_k =& -\mathcal{R}(X_k + \tau_k \alpha_{21} K_1) + \gamma_{21} E^T K_1 E \\
\tilde{A}_k^T K_3 E + E^T K_3 \tilde{A}_k =& -\mathcal{R}\left(X_k + \tau_k \sum_{j=1}^{2} \alpha_{3j} K_j\right) + E^T \left(\sum_{j=1}^{2} \gamma_{3j} K_j\right) E \\
\tilde{A}_k^T K_4 E + E^T K_4 \tilde{A}_k =& -\mathcal{R}\left(X_k + \tau \sum_{j=1}^{3} \alpha_{4j} K_j\right) + E^T \left(\sum_{j=1}^{3} \gamma_{4j} K_j\right) E
\end{aligned}
$$

with $\tilde{A}_k := \frac{1}{2}\left(\tau_k (A - BB^T X_k E) - E\right)$

Rosenbrock Methods

Higher Order Schemes

4^{th} order scheme (Ros4) [Shampine '82]

Determining Coefficients

$\alpha_{21} = 1$ $\alpha_{31} = \frac{24}{25}$ $\alpha_{41} = \frac{24}{25}$	$\alpha_{32} = \frac{3}{25}$ $\alpha_{42} = \frac{3}{25}$	$\alpha_{43} = 0$
$\gamma_{21} = 4$ $\gamma_{31} = -\frac{185}{125}$ $\gamma_{41} = \frac{56}{125}$	$\gamma_{32} = -\frac{6}{5}$ $\gamma_{42} = \frac{27}{125}$	$\gamma_{43} = \frac{1}{5}$
$b_1 = \frac{19}{18}$ $b_3 = \frac{25}{216}$	$b_2 = \frac{1}{4}$ $b_4 = \frac{125}{216}$	

with $\tilde{A}_k := \frac{1}{2}\left(\tau_k(A - BB^T X_k E) - E\right)$

Code Optimization and Implementation

Optimize before Parallelize

High Computational Cost

- Need the solution of s algebraic Lyapunov equations per time step.
  $\rightarrow$ Bartels-Stewart algorithm requires a QZ decomposition.
- Mostly matrix-matrix products.

Code Optimization and Implementation

Optimize before Parallelize

High Computational Cost

- Need the solution of s algebraic Lyapunov equations per time step.
  $\rightarrow$ Bartels-Stewart algorithm requires a QZ decomposition.
- Mostly matrix-matrix products.

Redundant Information

- Each right hand side of the Lyapunov equation includes $\mathcal{R}(X_k)$.
- Redundant information in the linear part of $\mathcal{R}(X_k + \tau_k K_j + \ldots)$.
- Solutions of the Lyapunov equations K_j are symmetric.

Code Optimization and Implementation

Optimize before Parallelize

High Computational Cost

- Need the solution of s algebraic Lyapunov equations per time step.
  $\rightarrow$ Bartels-Stewart algorithm requires a QZ decomposition.
- Mostly matrix-matrix products.

Redundant Information

- Each right hand side of the Lyapunov equation includes $\mathcal{R}(X_k)$.
- Redundant information in the linear part of $\mathcal{R}(X_k + \tau_k K_j + \ldots)$.
- Solutions of the Lyapunov equations K_j are symmetric.

Strategies

- Computational Cost: Use BLAS level-3 enabled algorithms.
- Redundant Information: Reformulation of the right hand sides.

Code Optimization and Implementation

BLAS level-3 enabled Algorithms

Matrix-Matrix Products

Use Intel® MKL, IBM ESSL, OpenBLAS or ATLAS.

Code Optimization and Implementation

BLAS level-3 enabled Algorithms

Matrix-Matrix Products

Use Intel® MKL, IBM ESSL, OpenBLAS or ATLAS.

Lyapunov Equations

Generalized Bartels-Stewart algorithm available in SLICOT: [Penzl '97]

- All stages have the same coefficient matrices.
- Only one QZ decomposition per time step and reuse it.

Code Optimization and Implementation

BLAS level-3 enabled Algorithms

Matrix-Matrix Products

Use Intel® MKL, IBM ESSL, OpenBLAS or ATLAS.

Lyapunov Equations

Generalized Bartels-Stewart algorithm available in SLICOT: [Penzl '97]

- All stages have the same coefficient matrices.
- Only one QZ decomposition per time step and reuse it.
- But QZ is mostly a BLAS level-2 algorithm.

Code Optimization and Implementation

BLAS level-3 enabled Algorithms

Matrix-Matrix Products

Use Intel® MKL, IBM ESSL, OpenBLAS or ATLAS.

Lyapunov Equations

Generalized Bartels-Stewart algorithm available in SLICOT: [Penzl '97]

- All stages have the same coefficient matrices.
- Only one QZ decomposition per time step and reuse it.
- But QZ is mostly a BLAS level-2 algorithm.
- Forward/Backward substitution is BLAS level-2 as well.

Code Optimization and Implementation

BLAS level-3 enabled Algorithms

Matrix-Matrix Products

Use Intel® MKL, IBM ESSL, OpenBLAS or ATLAS.

Lyapunov Equations

Generalized Bartels-Stewart algorithm available in SLICOT: [Penzl '97]

- All stages have the same coefficient matrices.
- Only one QZ decomposition per time step and reuse it.
- But QZ is mostly a BLAS level-2 algorithm.
- Forward/Backward substitution is BLAS level-2 as well.

Need for an efficient Lyapunov solver

Code Optimization and Implementation

BLAS level-3 enabled Algorithms

Matrix-Matrix Products

Use Intel® MKL, IBM ESSL, OpenBLAS or ATLAS.

Lyapunov Equations

Generalized Bartels-Stewart algorithm available in SLICOT: [Penzl '97]

- All stages have the same coefficient matrices.
- Only one QZ decomposition per time step and reuse it.
- But QZ is mostly a BLAS level-2 algorithm.
- Forward/Backward substitution is BLAS level-2 as well.

Need for an efficient Lyapunov solver

- Matrix Sign Function Iteration [Quintana-Ortí, Benner'99]
  $\rightarrow$ Without QZ decomposition, but no advantage out of coefficients.

Code Optimization and Implementation

BLAS level-3 enabled Algorithms

Matrix-Matrix Products

Use Intel® MKL, IBM ESSL, OpenBLAS or ATLAS.

Lyapunov Equations

Generalized Bartels-Stewart algorithm available in SLICOT: [Penzl '97]

- All stages have the same coefficient matrices.
- Only one QZ decomposition per time step and reuse it.
- But QZ is mostly a BLAS level-2 algorithm.
- Forward/Backward substitution is BLAS level-2 as well.

Need for an efficient Lyapunov solver

- Matrix Sign Function Iteration [Quintana-Ortí, Benner'99]
  $\rightarrow$ Without QZ decomposition, but no advantage out of coefficients.
- Reuse of the QZ decomposition and BLAS level-3 block generalized Bartels-Stewart algorithm. [GLYAP3: K., Saak '14]

Code Optimization and Implementation

Right-Hand-Side Rearrangement

Consider the stages of the 2^{nd} order Rosenbrock scheme:

$$\tilde{A}_k^T K_1 E + E^T K_1 \tilde{A}_k = -\mathcal{R}(X_k)$$

and

$$\tilde{A}_k^T K_2 E + E^T K_2 \tilde{A}_k = -\mathcal{R}(X_k + \tau_k K_1) + 2E^T K_1 E$$

with $\tilde{A} := \gamma\tau_k(A - BB^T X_k E) - \frac{1}{2}E$.

Code Optimization and Implementation

Right-Hand-Side Rearrangement

Consider the stages of the 2^{nd} order Rosenbrock scheme:

$$\tilde{A}_k^T K_1 E + E^T K_1 \tilde{A}_k = -\mathcal{R}(X_k)$$

and

$$\begin{aligned}
\tilde{A}_k^T K_2 E + E^T K_2 \tilde{A}_k &= -\mathcal{R}(X_k + \tau_k K_1) + 2E^T K_1 E \\
&= -\mathcal{R}(X_k) - \tau_k \left((A^T - BB^T X_k E)^T K_1 E + E^T K_1 (A^T - BB^T X_k E) \right) \\
&\quad + \tau_k^2 E^T K_1 BB^T K_1 E + 2E^T K_1 E
\end{aligned}$$

Code Optimization and Implementation

Right-Hand-Side Rearrangement

Consider the stages of the 2^{nd} order Rosenbrock scheme:

$$\tilde{A}_k^T K_1 E + E^T K_1 \tilde{A}_k = -\mathcal{R}(X_k)$$

and

$$\begin{aligned}
&\tilde{A}_k^T K_2 E + E^T K_2 \tilde{A}_k = -\mathcal{R}(X_k + \tau_k K_1) + 2E^T K_1 E \\
&= -\mathcal{R}(X_k) - \tau_k \left((A^T - BB^T X_k E)^T K_1 E + E^T K_1 (A^T - BB^T X_k E) \right) \\
&\quad + \tau_k^2 E^T K_1 BB^T K_1 E + 2E^T K_1 E \\
&= -\mathcal{R}(X_k) - \frac{1}{\gamma} \left(\tilde{A}_k^T K_1 E + E^T K_1 \tilde{A}_k \right) - \frac{1}{\gamma} E^T K_1 E \\
&\quad + \tau_k^2 E^T K_1 BB^T K_1 E + 2E^T K_1 E
\end{aligned}$$

Code Optimization and Implementation

Right-Hand-Side Rearrangement

Consider the stages of the 2^{nd} order Rosenbrock scheme:

$$\tilde{A}_k^T K_1 E + E^T K_1 \tilde{A}_k = -\mathcal{R}(X_k)$$

and

$$\begin{aligned}
&\tilde{A}_k^T K_2 E + E^T K_2 \tilde{A}_k = -\mathcal{R}(X_k + \tau_k K_1) + 2E^T K_1 E \\
&= -\mathcal{R}(X_k) - \tau_k \left((A^T - BB^T X_k E)^T K_1 E + E^T K_1 (A^T - BB^T X_k E) \right) \\
&\quad + \tau_k^2 E^T K_1 BB^T K_1 E + 2E^T K_1 E \\
&= -\mathcal{R}(X_k) - \frac{1}{\gamma} \left(\tilde{A}_k^T K_1 E + E^T K_1 \tilde{A}_k \right) - \frac{1}{\gamma} E^T K_1 E \\
&\quad + \tau_k^2 E^T K_1 BB^T K_1 E + 2E^T K_1 E \\
&= -(1 - \frac{1}{\gamma})\mathcal{R}(X_k) + \tau_k^2 E^T K_1 BB^T K_1 E + (2 - \frac{1}{\gamma}) E^T K_1 E.
\end{aligned}$$

Code Optimization and Implementation

Right-Hand-Side Rearrangement

Using the linearity of the Lyapunov-Equation we reformulate the 2^{nd} order Rosenbrock scheme as:

$$
\begin{aligned}
X_{k+1} &= X_k + \frac{3}{2}\tau_k K_1 + \frac{1}{2}\tau_k K_2, \\
\tilde{A}_k^T K_1 E + E^T K_1 \tilde{A}_k &= -\mathcal{R}(X_k), \\
\tilde{A}_k^T \tilde{K}_2 E + E^T \tilde{K}_2 \tilde{A}_k &= \tau_k^2 E^T K_1 B B^T K_1 E + (2 - \frac{1}{\gamma}) E^T K_1 E, \\
K_2 &= \tilde{K}_2 + (1 - \frac{1}{\gamma}) K_1.
\end{aligned}
$$

Code Optimization and Implementation

Right-Hand-Side Rearrangement

Using the linearity of the Lyapunov-Equation we reformulate the 2^{nd} order Rosenbrock scheme as:

$$X_{k+1} = X_k + \frac{3}{2}\tau_k K_1 + \frac{1}{2}\tau_k K_2,$$

$$\tilde{A}_k^T K_1 E + E^T K_1 \tilde{A}_k = -\mathcal{R}(X_k),$$

$$\tilde{A}_k^T \tilde{K}_2 E + E^T \tilde{K}_2 \tilde{A}_k = \tau_k^2 E^T K_1 B B^T K_1 E + (2 - \frac{1}{\gamma}) E^T K_1 E,$$

$$K_2 = \tilde{K}_2 + (1 - \frac{1}{\gamma}) K_1.$$

Code Optimization and Implementation

Right-Hand-Side Rearrangement

Using the linearity of the Lyapunov-Equation we reformulate the 2^{nd} order Rosenbrock scheme as:

$$
\begin{aligned}
X_{k+1} &= X_k + \frac{3}{2}\tau_k K_1 + \frac{1}{2}\tau_k K_2, \\
\tilde{A}_k^T K_1 E + E^T K_1 \tilde{A}_k &= -\mathcal{R}(X_k), \\
\tilde{A}_k^T \tilde{K}_2 E + E^T \tilde{K}_2 \tilde{A}_k &= \tau_k^2 E^T K_1 B B^T K_1 E + (2 - \frac{1}{\gamma}) E^T K_1 E, \\
K_2 &= \tilde{K}_2 + (1 - \frac{1}{\gamma}) K_1.
\end{aligned}
$$

- The 3^{rd} and 4^{th} order scheme can be rearranged in the same way.

Code Optimization and Implementation

Right-Hand-Side Rearrangement

Using the linearity of the Lyapunov-Equation we reformulate the 2^{nd} order Rosenbrock scheme as:

$$
\begin{aligned}
X_{k+1} &= X_k + \frac{3}{2}\tau_k K_1 + \frac{1}{2}\tau_k K_2, \\
\tilde{A}_k^T K_1 E + E^T K_1 \tilde{A}_k &= -\mathcal{R}(X_k), \\
\tilde{A}_k^T \tilde{K}_2 E + E^T \tilde{K}_2 \tilde{A}_k &= \tau_k^2 E^T K_1 B B^T K_1 E + (2 - \frac{1}{\gamma}) E^T K_1 E, \\
K_2 &= \tilde{K}_2 + (1 - \frac{1}{\gamma}) K_1.
\end{aligned}
$$

- The 3^{rd} and 4^{th} order scheme can be rearranged in the same way.
- Symmetric terms are computed like

$$
E^T K_j B B^T K_j E = {K_E^{(j)}}^T K_E^{(j)} \quad \text{with} \quad K_E^{(j)} := B^T K_j E.
$$

Code Optimization and Implementation

Parareal Implementation

Following [Maday, Lions, Turinici '01] we use

$$X_0^{(k+1)} := X(t = t_f),$$
$$X_p^{(k+1)} := F(t_{p-1}, t_p, X_{p-1}^{(k)}) + G(t_{p-1}, t_p, X_{p-1}^{(k+1)}) - G(t_{p-1}, t_p, X_{p-1}^{(k)})$$

as parareal-scheme, where $F(t_1, t_2, X_s)$ and $G(t_1, t_2, X_s)$ integrate the DRE from t_1 to t_2 with the initial value X_s.

Code Optimization and Implementation

Parareal Implementation

Following [Maday, Lions, Turinici '01] we use

$$X_0^{(k+1)} := X(t = t_f),$$
$$X_p^{(k+1)} := F(t_{p-1}, t_p, X_{p-1}^{(k)}) + G(t_{p-1}, t_p, X_{p-1}^{(k+1)}) - G(t_{p-1}, t_p, X_{p-1}^{(k)})$$

as parareal-scheme, where $F(t_1, t_2, X_s)$ and $G(t_1, t_2, X_s)$ integrate the DRE from t_1 to t_2 with the initial value X_s.

Coarse and Fine Solvers

Use the four presented Rosenbrock methods as coarse and the fine solvers:

- The coarse solver G performs one time step from t_1 to t_2.
- The fine solver F performs f time steps from t_1 to t_2.

Code Optimization and Implementation

Parareal Implementation

Classical Pipeline Implementation: Stage-Code

1: **for** `it:=1 to maxit` **do**
2: Receive $X_s^{(it)}$ from `ProcessID-1`
3: $X_G^{(it)} = G(X_s^{(it)})$
4: **if** $\texttt{it} = 1$ **then**
5: Send $X_G^{(it)}$ to `ProcessID+1`
6: **else**
7: $X^{(it)} := X_G^{(it)} + X_F^{(it-1)} - X_G^{(it-1)}$
8: **end if**
9: $X_F^{(it)} = F(X_s^{(it-1)})$
10: **if** $||X^{(it)} - X^{(it-1)}|| < \delta ||X^{(it)}||$ **then**
11: Stop.
12: **end if**
13: **end for**

Code Optimization and Implementation

Parareal Implementation

Classical Pipeline Implementation: Stage-Code

1: **for** `it:=1 to maxit` **do**
2: Receive $X_s^{(it)}$ from `ProcessID-1`
3: $X_G^{(it)} = G(X_s^{(it)})$
4: **if** $\texttt{it} = 1$ **then**
5: Send $X_G^{(it)}$ to `ProcessID+1`
6: **else**
7: $X^{(it)} := X_G^{(it)} + X_F^{(it-1)} - X_G^{(it-1)}$
8: **end if**
9: $X_F^{(it)} = F(X_s^{(it-1)})$
10: **if** $||X^{(it)} - X^{(it-1)}|| < \delta ||X^{(it)}||$ **then**
11: Stop.
12: **end if**
13: **end for**

Can be easily implemented on

- distributed systems using MPI,
- shared memory systems using OpenMP or PThreads.

Code Optimization and Implementation

Parareal Implementation

Classical Pipeline Implementation: Stage-Code

1: **for** `it:=1 to maxit` **do**
2: Receive $X_s^{(it)}$ from `ProcessID-1`
3: $X_G^{(it)} = G(X_s^{(it)})$
4: **if** $it = 1$ **then**
5: Send $X_G^{(it)}$ to `ProcessID+1`
6: **else**
7: $X^{(it)} := X_G^{(it)} + X_F^{(it-1)} - X_G^{(it-1)}$
8: **end if**
9: $X_F^{(it)} = F(X_s^{(it-1)})$
10: **if** $||X^{(it)} - X^{(it-1)}|| < \delta ||X^{(it)}||$ **then**
11: Stop.
12: **end if**
13: **end for**

Background Operation

Send is performed as a thread in background to be non-blocking.

MPI with threading-support required.

Can be easily implemented on

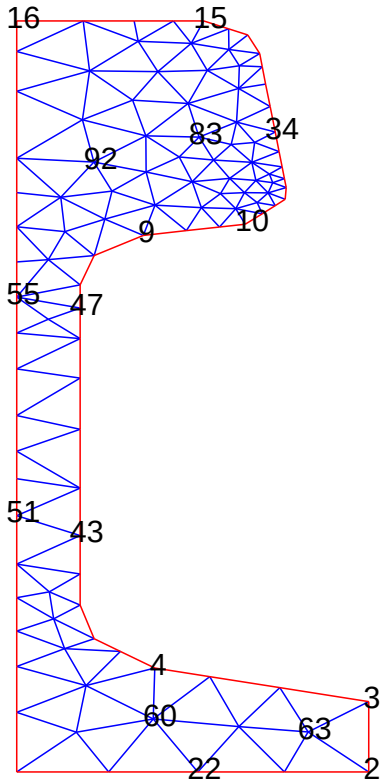
- distributed systems using MPI,
- shared memory systems using OpenMP or PThreads.

Experimental Results

Model Problem

- Mathematical model: boundary control for linearized 2D heat equation.

$$
\begin{aligned}
c \cdot \rho \frac{\partial}{\partial t} x &= \lambda \Delta x, && \xi \in \Omega \\
\lambda \frac{\partial}{\partial n} x &= \kappa(u_k - x), && \xi \in \Gamma_k,\ 1 \le k \le 7, \\
\frac{\partial}{\partial n} x &= 0, && \xi \in \Gamma_7.
\end{aligned}
$$



- FEM discretization with $n = 371$ states, $m = 7$ inputs, and $q = 6$ outputs
- computations with $\tau = 0.1ms$ on $[0, 45]s$
- evaluations for one component of the feedback $K(t) = -B^T X(t) E$

---

http://simulation.uni-freiburg.de/downloads/benchmark/Steel%20Profiles%20(38881)/

Experimental Results

Hardware Environment

HPC-Cluster otto

- Use 38 nodes $\equiv$ 456 Intel® Xeon® Westmere-EP cores @ 2.66GHz
  $\rightarrow$ only 450 cores used.
- 4 GB RAM per core
- QDR-Infiniband interconnect

Software

- Intel® Parallel Studio 2015 XE
- OpenMPI 1.8.1 with threading support.
- Intel® MKL 11.2.1

Experimental Results

Hardware Environment

HPC-Cluster otto

- Use 38 nodes $\equiv$ 456 Intel® Xeon® Westmere-EP cores @ 2.66GHz
  $\rightarrow$ only 450 cores used.
- 4 GB RAM per core
- QDR-Infiniband interconnect

Software

- Intel® Parallel Studio 2015 XE
- OpenMPI 1.8.1 with threading support.
- Intel® MKL 11.2.1

Reference Result

- 450 000 with $\tau = 0.1$ms with Ros4 in **3.46 days**.
- Storage for the trajectory: $X(t)$ – **1.4 TB**, $K(t)$ – **9 GB**

Experimental Results

Sequential Runtimes

Rosenbrock Order	SLICOT	GLYAP 3	ratio
1	4.40d	2.87d	1.53
2	6.15d	3.06d	2.01
3	7.84d	3.27d	2.40
4	9.55d	3.46d	2.75

Table: Sequential Runtime of the Rosenbrock methods with different Lyapunov solvers on a Westmere-EP CPU.

Experimental Results

Sequential Runtimes

Rosenbrock Order	SLICOT	GLYAP 3	ratio
1	4.40d	2.87d	1.53
2	6.15d	3.06d	2.01
3	7.84d	3.27d	2.40
4	9.55d	3.46d	2.75

Table: Sequential Runtime of the Rosenbrock methods with different Lyapunov solvers on a Westmere-EP CPU.

Rosenbrock Order	SLICOT	GLYAP 3	ratio
1	2.82d	1.79d	1.58
2	3.97d	1.91d	2.07
3	5.02d	1.97d	2.55
4	6.09d	2.07d	2.94

Table: Sequential Runtime of the Rosenbrock methods with different Lyapunov solvers on a Haswell-EP CPU (Intel® Xeon® E5-2640 v3 @ 2.60GHz).

Experimental Results

Sequential Runtimes

Rosenbrock Order	SLICOT	GLYAP 3	ratio
1	4.40d	2.87d	1.53
2	6.15d	3.06d	2.01
3	7.84d	3.27d	2.40
4	9.55d	3.46d	2.75

Table: Sequential Runtime of the Rosenbrock methods with different Lyapunov solvers on a Westmere-EP CPU

Optimize before parallelize already gains a speed up of 2.76
(or 2.94 on newer architectures).

Rosenbrock Order	SLICOT	GLYAP 3	ratio
1	2.82d	1.79d	1.58
2	3.97d	1.91d	2.07
3	5.02d	1.97d	2.55
4	6.09d	2.07d	2.94

Table: Sequential Runtime of the Rosenbrock methods with different Lyapunov solvers on a Haswell-EP CPU (Intel® Xeon® E5-2640 v3 @ 2.60GHz).

Experimental Results

Distributed Parallel Execution

Parareal Setup

- 450 coarse steps, $\tau_{coarse} = 100\text{ms}$
- 1000 fine steps per coarse step, $\tau = 0.1\text{ms}$
- Maximum number of iterations: 10
- Cancellation criteria: $\delta = 10^{-6}$

Experimental Results

Distributed Parallel Execution

Parareal Setup

- 450 coarse steps, $\tau_{coarse} = 100$ms
- 1000 fine steps per coarse step, $\tau = 0.1$ms
- Maximum number of iterations: 10
- Cancellation criteria: $\delta = 10^{-6}$

Fine \ Coarse	Ros1		Ros2		Ros3		Ros4	
	Time	It	Time	It	Time	It	Time	It
Ros1	3.30h	9	2.97h	8	3.29h	9	1.74h	4
Ros2	3.59h	9	3.27h	8	3.61h	9	1.89h	4
Ros3	3.87h	9	3.51h	8	3.88h	9	2.02h	4
Ros4	4.17h	9	3.78h	8	4.18h	9	2.19h	4

Table: Runtime and maximum iteration number.

Experimental Results

Distributed Parallel Execution

Parareal Setup

- 450 coarse steps, $\tau_{coarse} = 100\text{ms}$
- 1000 fine steps per coarse step, $\tau = 0.1\text{ms}$
- Maximum number of iterations: 10
- Cancellation criteria: $\delta = 10^{-6}$

Fine \ Coarse	Ros1	Ros2	Ros3	Ros4
Ros1	2.01e-05	2.01e-05	2.01e-05	2.01e-05
Ros2	2.07e-05	2.07e-05	2.07e-05	2.07e-05
Ros3	2.07e-05	2.07e-05	2.07e-05	2.07e-05
Ros4	1.27e-09	3.00e-10	9.71e-08	6.10e-14

Table: Relative 1-norm error between the Parareal solution and the reference.

Experimental Results

Distributed Parallel Execution

Parareal Setup

- 450 coarse steps, $\tau_{coarse} = 100\text{ms}$
- 1000 fine steps per coarse step, $\tau = 0.1\text{ms}$
- Maximum number of iterations: 10
- Cancellation criteria: $\delta = 10^{-6}$

Fine \ Coarse	Ros1	Ros2	Ros3	Ros4
Ros1	*	7.69e-05	7.68e-05	7.68e-05
Ros2	*	7.81e-05	7.80e-05	7.80e-05
Ros3	*	7.81e-05	7.80e-05	7.80e-05
Ros4	*	3.14e-11	5.76e-09	1.14e-14

Table: Relative 1-norm error between the Parareal solution and the reference for $K(t)_{1,77}$.

Experimental Results

Pipeline View

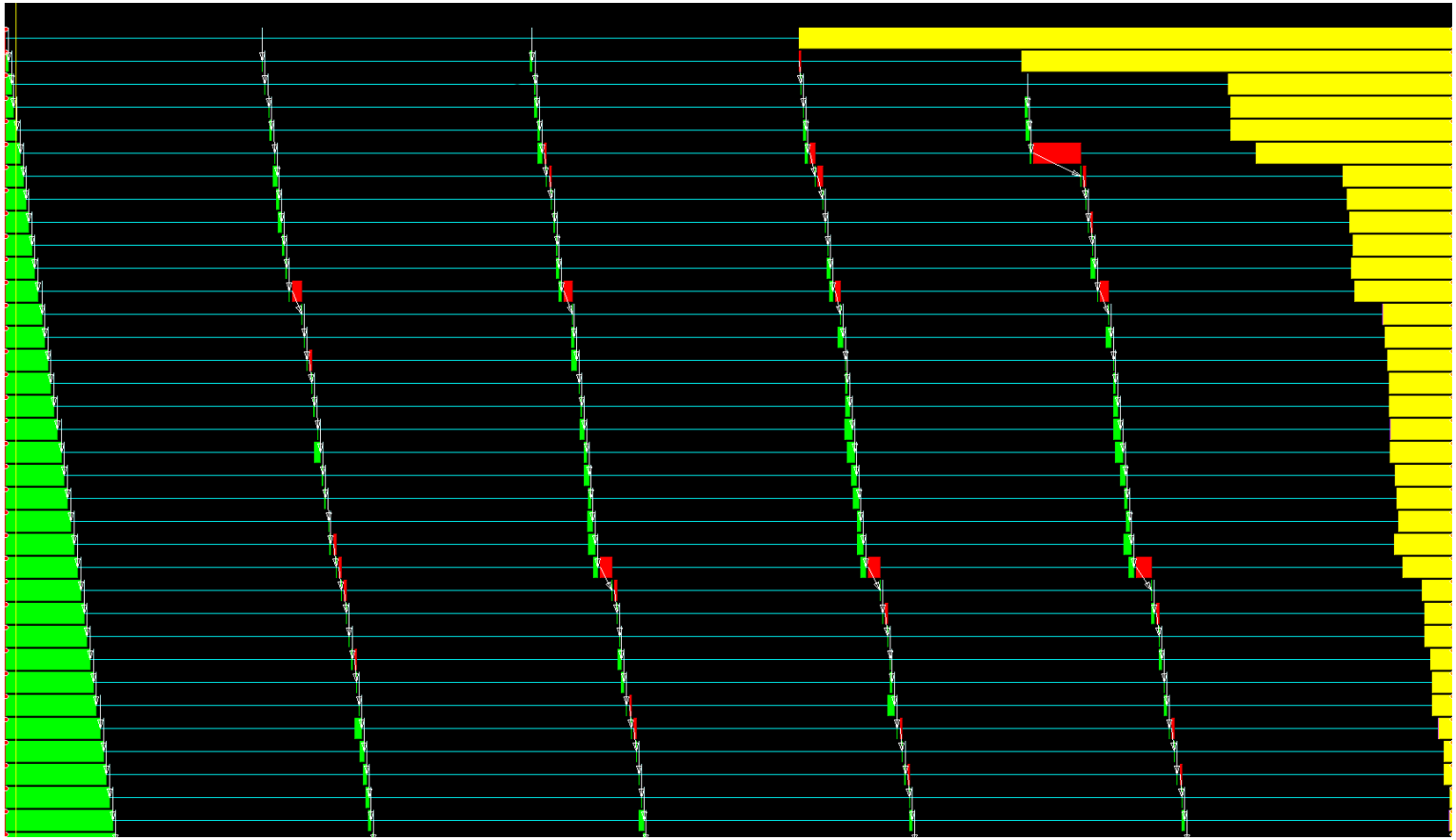


Figure: Pipeline view for 36 coarse, 100 fine steps.

Conclusions and Open Problems

Conclusions

We have seen that:

- "optimize before parallelize" already gains nearly a factor of up to 3,
- Parareal shrinks the runtime down to 2.19h with accurate results.

Conclusions and Open Problems

Conclusions

We have seen that:

- "optimize before parallelize" already gains nearly a factor of up to 3,
- Parareal shrinks the runtime down to 2.19h with accurate results.

We observed that:

- (our) Parareal implementation requires the same computational complexity for each evaluation of the coarse or the fine solver,
- we have relatively long startup phase until all processors work in parallel,
- we are restricted to one-step methods on the coarse level.

Conclusions and Open Problems

Conclusions

We have seen that:

- "optimize before parallelize" already gains nearly a factor of up to 3,
- Parareal shrinks the runtime down to 2.19h with accurate results.

We observed that:

- (our) Parareal implementation requires the same computational complexity for each evaluation of the coarse or the fine solver,
- we have relatively long startup phase until all processors work in parallel,
- we are restricted to one-step methods on the coarse level.

Preliminary experiments on Intel® Xeon® Phi showed:

- pipeline startup takes too long due to the poor sequential performance evaluating the coarse solver,
- Even the 1st order Rosenbrock method is too expensive here,
- Card memory can only hold the feedback matrix $K(t) = B^T X(t) E$.

Conclusions and Open Problems

Open Problems

Step-size control accelerates the sequential code. How to build and step-size control aware Parareal scheme?

- How to distribute the time grid?
- How to setup the pipeline processing?
- How to get a good load balancing?

Conclusions and Open Problems

Open Problems

Step-size control accelerates the sequential code. How to build and step-size control aware Parareal scheme?

- How to distribute the time grid?
- How to setup the pipeline processing?
- How to get a good load balancing?

BDF schemes are a good alternative to the Rosenbrock schemes.

- BDF as fine solver inside a pipeline stage: no problem!
- How to use BDF as coarse solver across the coarse time grid?

Conclusions and Open Problems

Open Problems

Step-size control accelerates the sequential code. How to build and step-size control aware Parareal scheme?

- How to distribute the time grid?
- How to setup the pipeline processing?
- How to get a good load balancing?

BDF schemes are a good alternative to the Rosenbrock schemes.

- BDF as fine solver inside a pipeline stage: no problem!
- How to use BDF as coarse solver across the coarse time grid?

For large scale DREs we have to approximate $X(t)$ by low rank factors $X(t) \approx Z(t)Z(t)^T$.

- Computational complexity of each call to the coarse and the fine solver differs. $\nearrow$ Load Balancing.
- The size of the low-rank factor $Z(t)$ differs in every step. $\nearrow$ Parareal formulation with low-rank factors.

Conclusions and Open Problems

Open Problems

Step-size control accelerates the sequential code. How to build and step-size control aware Parareal scheme?

- How to distribute the time grid?
- How to setup the pipeline processing?
- How to get a good load balancing?

Thank you for your attention! Questions?

- BDF as fine solver inside a pipeline stage: no problem!
- How to use BDF as coarse solver across the coarse time grid?

For large scale DREs we have to approximate $X(t)$ by low rank factors $X(t) \approx Z(t)Z(t)^T$.

- Computational complexity of each call to the coarse and the fine solver differs. $\nearrow$ Load Balancing.
- The size of the low-rank factor $Z(t)$ differs in every step. $\nearrow$ Parareal formulation with low-rank factors.