An introduction to fractional calculus

Fundamental ideas and numerics

Fabio Durastante
Università di Pisa
fabio.durastante@unipi.it
fdurastante.github.io

October, 2022

All-at-once

We have seen that for a problem of the form

$$\begin{cases} u_t = \mathcal{L}(u), & u \,:\, \Omega \times [0,T] \to \mathbb{R}^d,\ \Omega \subseteq \mathbb{R}^d \\ u(\mathbf{x},0) = u_0(\mathbf{x}), & \\ \mathcal{B}(u) = 0, & \mathbf{x} \in \partial\Omega. \end{cases}$$

with

- $\mathcal{L}(\cdot)$ a *linear* and *autonomous* differential operator (possibly involving fractional derivatives),
- or changing u_t with ${}^{CA}D^{\alpha}_{[0,t]}u$,

we can **rewrite it** as a single linear system/matrix equation.

All-at-once

We have seen that for a problem of the form

$$\begin{cases} \mathbf{u}_t = \mathcal{L}_h(\mathbf{u}), & \mathbf{u} : \mathbb{R}^n \times [0, T] \to \mathbb{R}^n \\ \mathbf{u}(0) = \mathbf{u}_0, \\ \mathcal{B}_h(\mathbf{u}) = 0. \end{cases}$$

with

- $\mathcal{L}(\cdot)$ a *linear* and *autonomous* differential operator (possibly involving fractional derivatives),
- or changing u_t with ${}^{CA}D^{\alpha}_{[0,t]}u$,

we can **rewrite it** as a single linear system/matrix equation.

To *abstract the procedure* let's think about working the **M**ethod **O**f **L**ine way!

All-at-once: system of autonomous ODE

Following the MOL trail, we now have to solve a system of autonomous ODEs:

$$M\mathbf{u}_t(t) = L\mathbf{u}(t), \qquad M, L \in \mathbb{R}^{n\times n},$$

that could be a **d**ifferential-**al**gebraic system of **e**quations (DAE) if $\det(M) = 0$.

All-at-once: system of autonomous ODE

Following the MOL trail, we now have to solve a system of autonomous ODEs:

$$M\mathbf{u}_t(t) = L\mathbf{u}(t), \qquad M, L \in \mathbb{R}^{n\times n},$$

- that could be a **d**ifferential-**al**gebraic system of **e**quations (DAE) if $\det(M) = 0$.
- To formulate the *all-at-once* procedure, one has to select a method to *march in time* the solution:

All-at-once: system of autonomous ODE

Following the MOL trail, we now have to solve a system of autonomous ODEs:

$$M\mathbf{u}_t(t) = L\mathbf{u}(t), \qquad M, L \in \mathbb{R}^{n \times n},$$

- that could be a **d**ifferential-**al**gebraic system of **e**quations (DAE) if $\det(M) = 0$.
- To formulate the *all-at-once* procedure, one has to select a method to *march in time* the solution:
  - Linear multistep methods,
  - Runge-Kutta methods,
  - General linear methods (a mix of the two above strategies).

Linear Multistep Methods

Given a general ODE of the form

$$u'(t) = f(t, u(t)), \quad u(t_0) = u_0,$$

a k-step LMM is a recursion of the form with step-size $h = t_{n+k} - t_{n+k-1} > 0$

$$\sum_{j=0}^{k} \alpha_j u_{n+j} = \sum_{j=0}^{k} h\beta_j f_{n+j}, \qquad f_m \triangleq f(t_m, y_m),$$

with coefficients $\alpha_j \in \mathbb{R}$ and $\beta_j \in \mathbb{R}$ $(j = 0, \ldots, k)$, and we are **interested only** in **implicit methods**, i.e., $\beta_k \neq 0$.

Linear Multistep Methods

Given a general ODE of the form

$$u'(t) = f(t, u(t)), \quad u(t_0) = u_0,$$

a k-step LMM is a recursion of the form with step-size $h = t_{n+k} - t_{n+k-1} > 0$

$$\sum_{j=0}^{k} \alpha_j u_{n+j} = \sum_{j=0}^{k} h\beta_j f_{n+j}, \qquad f_m \triangleq f(t_m, y_m),$$

with coefficients $\alpha_j \in \mathbb{R}$ and $\beta_j \in \mathbb{R}$ $(j = 0, \ldots, k)$, and we are **interested only** in **implicit methods**, i.e., $\beta_k \neq 0$.
They can be analyzed by looking at the **polynomials**

$$\rho(\zeta) = \sum_{j=0}^{k} \alpha_j \zeta^j = (\zeta - 1) \sum_{j=0}^{k-1} \gamma_j \zeta^j = (\zeta - 1) \cdot \rho_R(\zeta), \qquad \sigma(\zeta) = \sum_{j=0}^{k} \beta_j \zeta^j.$$

Linear Multistep Methods

0-stable method

A method is 0-stable if all roots of $\rho(\zeta) = (\zeta - 1) \cdot \rho_R(\zeta) = 0$ lie inside or on the unit circle, with no multiple unimodular roots.

- *Zero stability* is necessary for convergence,
- It is a condition on the *extraneous operator* $\rho_R(\zeta)$, i.e., a condition on the k coefficients $\{\gamma_j\}_{j=0}^{k-1}$.

Linear Multistep Methods

0-stable method

A method is 0-stable if all roots of $\rho(\zeta) = (\zeta - 1) \cdot \rho_R(\zeta) = 0$ lie inside or on the unit circle, with no multiple unimodular roots.

- *Zero stability* is necessary for convergence,
- It is a condition on the *extraneous operator* $\rho_R(\zeta)$, i.e., a condition on the k coefficients $\{\gamma_j\}_{j=0}^{k-1}$.

A-stable method

The behavior of these methods can be analyzed by applying them on the test problem $y' = ky$ subject to the initial condition $y(0) = 1$ with $k \in \mathbb{C}$. The solution of this equation is $y(t) = e^{kt}$. If the numerical method exhibits the same behavior of the solution for a fixed step size, then the method is said to be A-stable.

Linear Multistep Methods

0-stable method

A method is 0-stable if all roots of $\rho(\zeta) = (\zeta - 1) \cdot \rho_R(\zeta) = 0$ lie inside or on the unit circle, with no multiple unimodular roots.

- *Zero stability* is necessary for convergence,
- It is a condition on the *extraneous operator* $\rho_R(\zeta)$, i.e., a condition on the k coefficients $\{\gamma_j\}_{j=0}^{k-1}$.

***A*-stable method**

The behavior of these methods can be analyzed by applying them on the test problem $y' = ky$ subject to the initial condition $y(0) = 1$ with $k \in \mathbb{C}$. The solution of this equation is $y(t) = e^{kt}$. If the numerical method exhibits the same behavior of the solution for a fixed step size, then the method is said to be A-stable.

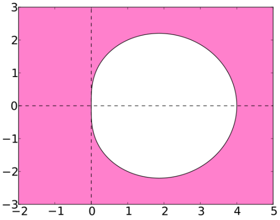
- Usually one ends up with limitations involving the admissible h.

Linear Multistep Methods: initial values

If we use a LMM with $k > 1$ we need more starting values than the one we have!

We are interested in **diffusion dominated problems**, thus **B**ackward-**D**ifferentiation **F**ormulas are a common choice.

	$\{\alpha_k\}_k,\ \beta_k = 1,\ \beta_j = 0,\ j \le k$						
BDF2					$1/2$	-2	$3/2$
BDF3				$-1/3$	$3/2$	-3	$11/6$
BDF4			$1/4$	$-4/3$	3	-4	$25/12$
BDF5		$-1/5$	$5/4$	$-10/3$	5	-5	$137/60$
BDF6	$1/6$	$-6/5$	$15/4$	$-20/3$	$15/2$	-6	$147/60$



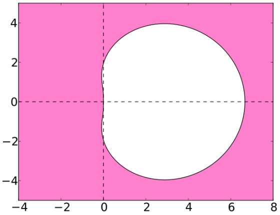
Methods with $k > 6$ are not zero-stable so they cannot be used.

Linear Multistep Methods: initial values

If we use a LMM with $k > 1$ we need more starting values than the one we have!

We are interested in **diffusion dominated problems**, thus **B**ackward-**D**ifferentiation **F**ormulas are a common choice.

$\{\alpha_k\}_k,\ \beta_k = 1,\ \beta_j = 0,\ j \leq k$							
BDF2					$1/2$	-2	$3/2$
BDF3				$-1/3$	$3/2$	-3	$11/6$
BDF4			$1/4$	$-4/3$	3	-4	$25/12$
BDF5		$-1/5$	$5/4$	$-10/3$	5	-5	$137/60$
BDF6	$1/6$	$-6/5$	$15/4$	$-20/3$	$15/2$	-6	$147/60$



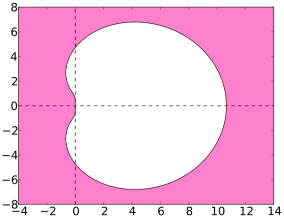
- Methods with $k > 6$ are not zero-stable so they cannot be used.
- If we want to use BDF6 we need 5 initial conditions, and have only one.
- We can use lower order BDFs to generate the step we need.

Linear Multistep Methods: initial values

If we use a LMM with $k > 1$ we need more starting values than the one we have!

We are interested in **diffusion dominated problems**, thus **B**ackward-**D**ifferentiation **F**ormulas are a common choice.

$\{\alpha_k\}_k,\ \beta_k = 1,\ \beta_j = 0,\ j \leq k$							
BDF2					$1/2$	-2	$3/2$
BDF3				$-1/3$	$3/2$	-3	$11/6$
BDF4			$1/4$	$-4/3$	3	-4	$25/12$
BDF5		$-1/5$	$5/4$	$-10/3$	5	-5	$137/60$
BDF6	$1/6$	$-6/5$	$15/4$	$-20/3$	$15/2$	-6	$147/60$



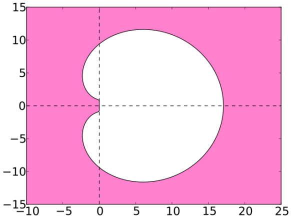
- Methods with $k > 6$ are not zero-stable so they cannot be used.
- If we want to use BDF6 we need 5 initial conditions, and have only one.
- We can use lower order BDFs to generate the step we need.

Linear Multistep Methods: initial values

If we use a LMM with $k > 1$ we need more starting values than the one we have!

We are interested in **diffusion dominated problems**, thus **B**ackward-**D**ifferentiation **F**ormulas are a common choice.

	$\{\alpha_k\}_k,\ \beta_k = 1,\ \beta_j = 0,\ j \leq k$						
BDF2					1/2	−2	3/2
BDF3				−1/3	3/2	−3	11/6
BDF4			1/4	−4/3	3	−4	25/12
BDF5		−1/5	5/4	−10/3	5	−5	137/60
BDF6	1/6	−6/5	15/4	−20/3	15/2	−6	147/60



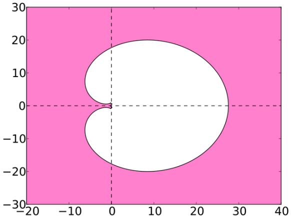
- Methods with $k > 6$ are not zero-stable so they cannot be used.
- If we want to use BDF6 we need 5 initial conditions, and have only one.
- We can use lower order BDFs to generate the step we need.

Linear Multistep Methods: initial values

If we use a LMM with $k > 1$ we need more starting values than the one we have!

We are interested in **diffusion dominated problems**, thus **B**ackward-**D**ifferentiation **F**ormulas are a common choice.

$\{\alpha_k\}_k,\ \beta_k = 1,\ \beta_j = 0,\ j \leq k$							
BDF2					$1/2$	-2	$3/2$
BDF3				$-1/3$	$3/2$	-3	$11/6$
BDF4			$1/4$	$-4/3$	3	-4	$25/12$
BDF5		$-1/5$	$5/4$	$-10/3$	5	-5	$137/60$
BDF6	$1/6$	$-6/5$	$15/4$	$-20/3$	$15/2$	-6	$147/60$



- Methods with $k > 6$ are not zero-stable so they cannot be used.
- If we want to use BDF6 we need 5 initial conditions, and have only one.
- We can use lower order BDFs to generate the step we need.

Linear Multistep Methods

From what we have seen in the last lectures we can write down the problem as

$$(A_m \otimes M_n - hB_m \otimes L_n)\mathbf{u} = \mathbf{f},$$

Linear Multistep Methods

From what we have seen in the last lectures we can write down the problem as

$$(\color{red}{A_m} \otimes M_n - hB_m \otimes L_n)\mathbf{u} = \mathbf{f},$$

$$A_m = \begin{bmatrix} 1 & & & & & & & \\ -2 & 3/2 & & & & & & \\ 3/2 & -3 & 11/6 & & & & & \\ -4/3 & 3 & -4 & 25/12 & & & & \\ 5/4 & -10/3 & 5 & -5 & 137/60 & & & \\ -6/5 & 15/4 & -20/3 & 15/2 & -6 & 147/60 & & \\ 1/6 & -6/5 & 15/4 & -20/3 & 15/2 & -6 & 147/60 & \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \end{bmatrix}_{(m-1)\times(m-1)}$$

Linear Multistep Methods

From what we have seen in the last lectures we can write down the problem as

$$(A_m \otimes M_n - hB_m \otimes L_n)\mathbf{u} = \mathbf{f},$$

$$B_m = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix}_{(m-1)\times(m-1)}$$

Linear Multistep Methods

From what we have seen in the last lectures we can write down the problem as

$$(A_m \otimes M_n - hB_m \otimes L_n)\mathbf{u} = \mathbf{f},$$

$$\mathbf{f} = \begin{bmatrix} \mathbf{u}_0 + f(t_1) \\ -{}^1\!/_2\mathbf{u}_0 + f(t_2) \\ {}^1\!/_3\mathbf{u}_0 + f(t_3) \\ -{}^1\!/_4\mathbf{u}_0 + f(t_4) \\ {}^1\!/_5\mathbf{u}_0 + f(t_5) \\ -{}^1\!/_6\mathbf{u}_0 + f(t_6) \\ f(t_7) \\ \vdots \end{bmatrix}$$

A simple example

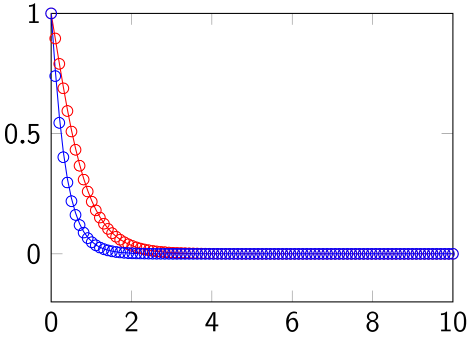
```
L = [-2, 1; 0, -3]; % Problem
y0 = [1;1];
n = length(L);
% Discretize
m = 100;
T = linspace(0,10,m); h = T(2)-T(1);
r = zeros(m-1,1); c = zeros(m-1,1);
r(1:7)=[147/60,-6,15/2,-20/3,15/4,-6/5,1/6];
c(1) = 147/60;
A = toeplitz(r,c);
A(1,1) = 1; % Fix BCs
A(2,1) = -2; A(2,2) = 3/2;
A(3,1) = 3/2; A(3,2) = -3; A(3,3) = 11/6;
A(4,1) = -4/3; A(4,2) = 3; A(4,3) = -4;
↪ A(4,4) = 25/12;
A(5,1) = 5/4; A(5,2) = -10/3; A(5,3) = 5;
```

```
A(5,4) = -5; A(5,5) = 137/60;
In = speye(n,n);
Im = speye(m-1,m-1);
%% Build rhs:
b = zeros((m-1)*n,1);
b(1:2) = y0;
b(3:4) = -1/2*y0;
b(5:6) = 1/3*y0;
b(7:8) = -1/4*y0;
b(9:10) = 1/5*y0;
b(11:12) = -1/6*y0;
% SOLVE (Linear system)
M = kron(A,In)-h*kron(Im,L);
x = M\b;
```

A simple example

We can compare the solution with `ode15s`, and visualize it

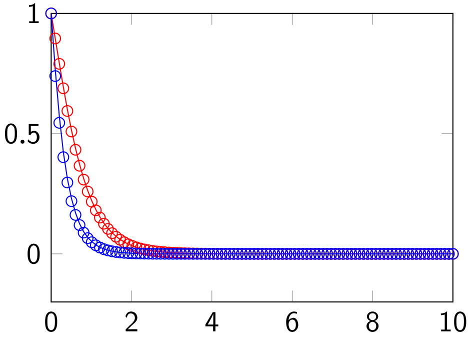
```
[tt,yy] = ode15s(@(t,y) L*y,T,y0);
X = reshape(x,n,m-1);
X = [y0,X];
% Plot
plot(T,X(1,:),'r-',T,X(2,:),'b-',...
T,yy(:,1),'ro',...
T,yy(:,2),'bo');
```



A simple example

We can compare the solution with ode15s, and visualize it

```
[tt,yy] = ode15s(@(t,y) L*y,T,y0);
X = reshape(x,n,m-1);
X = [y0,X];
% Plot
plot(T,X(1,:),'r-',T,X(2,:),'b-',...
T,yy(:,1),'ro',...
T,yy(:,2),'bo');
```

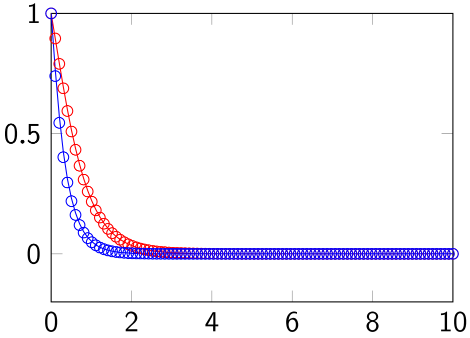


We could solve everything using a matrix-equation based solver,

A simple example

We can compare the solution with `ode15s`, and visualize it

```
[tt,yy] = ode15s(@(t,y) L*y,T,y0);
X = reshape(x,n,m-1);
X = [y0,X];
% Plot
plot(T,X(1,:),'r-',T,X(2,:),'b-',...
T,yy(:,1),'ro',...
T,yy(:,2),'bo');
```

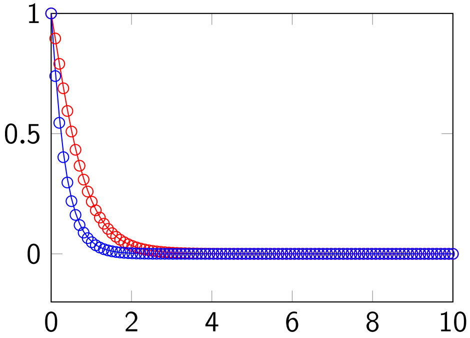


- We could solve everything using a matrix-equation based solver,
- but we are looking at a case in which $m = 2$ with a “non refinable” space operator.

A simple example

We can compare the solution with ode15s, and visualize it

```
[tt,yy] = ode15s(@(t,y) L*y,T,y0);
X = reshape(x,n,m-1);
X = [y0,X];
% Plot
plot(T,X(1,:),'r-',T,X(2,:),'b-',...
T,yy(:,1),'ro',...
T,yy(:,2),'bo');
```



- We could solve everything using a matrix-equation based solver,
- but we are looking at a case in which $m = 2$ with a "non refinable" space operator.
- What can we say about the A_m matrix?

Matrix properties

A_m is a banded Toeplitz matrix plus a rank correction.

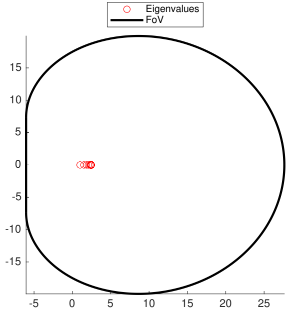
$$A_m = \begin{bmatrix} 1 & & & & & & & \\ -2 & 3/2 & & & & & & \\ 3/2 & -3 & 11/6 & & & & & \\ -4/3 & 3 & -4 & 25/12 & & & & \\ 5/4 & -10/3 & 5 & -5 & 137/60 & & & \\ -6/5 & 15/4 & -20/3 & 15/2 & -6 & 147/60 & & \\ 1/6 & -6/5 & 15/4 & -20/3 & 15/2 & -6 & 147/60 & \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \end{bmatrix}$$

- We know the eigenvalues in closed form: it's lower triangular!

Matrix properties

A_m is a banded Toeplitz matrix plus a rank correction.

$$A_m = \begin{bmatrix} 1 & & & & & & & \\ -2 & 3/2 & & & & & & \\ 3/2 & -3 & 11/6 & & & & & \\ -4/3 & 3 & -4 & 25/12 & & & & \\ 5/4 & -10/3 & 5 & -5 & 137/60 & & & \\ -6/5 & 15/4 & -20/3 & 15/2 & -6 & 147/60 & & \\ 1/6 & -6/5 & 15/4 & -20/3 & 15/2 & -6 & 147/60 & \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \end{bmatrix}$$

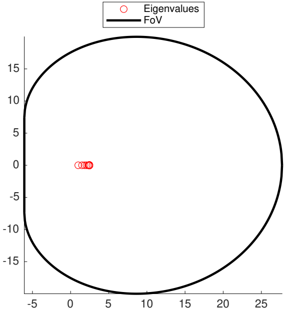


- We know the eigenvalues in closed form: it's lower triangular!
- The Field-Of-Values contains the origin... bad for bounds!

Matrix properties

A_m is a banded Toeplitz matrix plus a rank correction.

$$A_m = \begin{bmatrix} 1 & & & & & & & \\ -2 & 3/2 & & & & & & \\ 3/2 & -3 & 11/6 & & & & & \\ -4/3 & 3 & -4 & 25/12 & & & & \\ 5/4 & -10/3 & 5 & -5 & 137/60 & & & \\ -6/5 & 15/4 & -20/3 & 15/2 & -6 & 147/60 & & \\ 1/6 & -6/5 & 15/4 & -20/3 & 15/2 & -6 & 147/60 & \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \end{bmatrix}$$

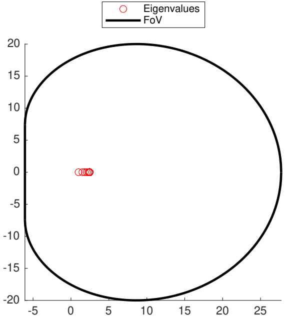


- We know the eigenvalues in closed form: it's lower triangular!
- The Field-Of-Values contains the origin... bad for bounds!

Matrix properties

A_m is a banded Toeplitz matrix plus a rank correction.

$$A_m = \begin{bmatrix} 1 \\ -2 & 3/2 \\ 3/2 & -3 & 11/6 \\ -4/3 & 3 & -4 & 25/12 \\ 5/4 & -10/3 & 5 & -5 & 137/60 \\ -6/5 & 15/4 & -20/3 & 15/2 & -6 & 147/60 \\ 1/6 & -6/5 & 15/4 & -20/3 & 15/2 & -6 & 147/60 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \end{bmatrix}$$



- We know the eigenvalues in closed form: it's lower triangular!
- The Field-Of-Values contains the origin… bad for bounds!
- Its clearly non diagonalizable, if we try and look at the condition number of the eigenvector matrix $\kappa_2(X_{100}) = 7.30 \times 10^{111}$.

Matrix properties

Indeed, already for the BDF1 (a.k.a. the implicit Euler method) we have

$$A_m = \begin{bmatrix} 1 & & & \\ -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{bmatrix}_{(m-1)\times(m-1)}$$

- It is a Jordan block, so *no diagonalization*,

Matrix properties

Indeed, already for the BDF1 (a.k.a. the implicit Euler method) we have

$$A_m = \begin{bmatrix} 1 & & & \\ -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{bmatrix}_{(m-1)\times(m-1)}$$

- It is a Jordan block, so *no diagonalization*,
- What do we expect for the matrix equation solver?

Matrix properties

Indeed, already for the BDF1 (a.k.a. the implicit Euler method) we have

$$A_m = \begin{bmatrix} 1 & & & \\ -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{bmatrix}_{(m-1)\times(m-1)}$$

- It is a Jordan block, so *no diagonalization*,
- What do we expect for the matrix equation solver?

BDF1, $\alpha = 1.5$

m	n	IT	Residual
64	128	13	1.007848e-10
128	256	16	6.145733e-10
256	512	21	7.639171e-10
512	1024	27	5.857467e-10
1024	2048	34	8.065585e-10
2048	4096	42	9.819085e-10

Matrix properties

Indeed, already for the BDF1 (a.k.a. the implicit Euler method) we have

$$A_m = \begin{bmatrix} 1 & & & \\ -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{bmatrix}_{(m-1)\times(m-1)}$$

- It is a Jordan block, so *no diagonalization*,
- What do we expect for the matrix equation solver?

BDF1, $\alpha = 1.5$

m	n	IT	Residual
64	128	13	1.007848e-10
128	256	16	6.145733e-10
256	512	21	7.639171e-10
512	1024	27	5.857467e-10
1024	2048	34	8.065585e-10
2048	4096	42	9.819085e-10

BDF6, $\alpha = 1.5$

m	n	IT	Residual
64	128	21	3.651570e-10
128	256	33	1.746513e-10
256	512	71	2.530720e-15
512	1024	128	1.975160e-22
1024	2048	251	4.157259e-10
2048	4096	495	6.310887e-10

Matrix properties

Indeed, already for the BDF1 (a.k.a. the implicit Euler method) we have

$$A_m = \begin{bmatrix} 1 & & & \\ -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{bmatrix}_{(m-1)\times(m-1)}$$

- It is a Jordan block, so *no diagonalization*,
- What do we expect for the matrix equation solver? Nothing good!

BDF1, $\alpha = 1.5$

m	n	IT	Residual
64	128	13	1.007848e-10
128	256	16	6.145733e-10
256	512	21	7.639171e-10
512	1024	27	5.857467e-10
1024	2048	34	8.065585e-10
2048	4096	42	9.819085e-10

BDF6, $\alpha = 1.5$

m	n	IT	Residual
64	128	21	3.651570e-10
128	256	33	1.746513e-10
256	512	71	2.530720e-15
512	1024	128	1.975160e-22
1024	2048	251	4.157259e-10
2048	4096	495	6.310887e-10

Linear Multistep Methods in Boundary Value Form

❓ Part of the problem, are those triangular matrices. Can we do something?

Linear Multistep Methods in Boundary Value Form

❓ Part of the problem, are those triangular matrices. Can we do something?

🔨 If we use *more than one step*, we still need **auxiliary formulas** to close the iteration.

Linear Multistep Methods in Boundary Value Form

- Part of the problem, are those triangular matrices. Can we do something?
- If we use *more than one step*, we still need **auxiliary formulas** to close the iteration.
- We could distribute the conditions differently, that is, not all on the initial data.

$$\sum_{j=-\nu}^{\mu-\nu} \alpha_{j+\nu}\mathbf{u}_{n+j} = h \sum_{j=-\nu}^{\mu-\nu} \beta_{j+\nu}\mathbf{f}_{n+j}, \quad n = \nu, \ldots, m-k+\nu.$$

- k steps,
- ν initial conditions, and
- $\mu - \nu$ final conditions,
- Described by $\rho(z) = z^{\nu} \sum_{j=-\nu}^{k-\nu} \alpha_{j+\nu} z^{j}$, and $\sigma(z) = z^{\nu} \sum_{j=-\nu}^{k-\nu} \beta_{j+\nu} z^{j}$.

Linear Multistep Methods in Boundary Value Form

- Part of the problem, are those triangular matrices. Can we do something?
- If we use *more than one step*, we still need **auxiliary formulas** to close the iteration.
- We could distribute the conditions differently, that is, not all on the initial data.

$$\sum_{j=-\nu}^{\mu-\nu} \alpha_{j+\nu} \mathbf{u}_{n+j} = h \sum_{j=-\nu}^{\mu-\nu} \beta_{j+\nu} \mathbf{f}_{n+j}, \quad n = \nu, \ldots, m-k+\nu.$$

  - k steps,
  - ν initial conditions, and
  - $\mu - \nu$ final conditions,
  - Described by $\rho(z) = z^\nu \sum_{j=-\nu}^{k-\nu} \alpha_{j+\nu} z^j$, and $\sigma(z) = z^\nu \sum_{j=-\nu}^{k-\nu} \beta_{j+\nu} z^j$.
  - How does this change matrices and stability?

Linear Multistep Methods in Boundary Value Form

If we collect the matrices for the inner steps of a *scalar* ODE, we get

$$A_m = \begin{bmatrix} \alpha_\nu & \cdots & \alpha_k & & & \\ \vdots & \ddots & & \ddots & & \\ \alpha_0 & & \ddots & & \ddots & \\ & \ddots & & \ddots & & \alpha_k \\ & & \ddots & & \ddots & \vdots \\ & & & \alpha_0 & \cdots & \alpha_\nu \end{bmatrix}_{(m-\nu)\times(m-\nu)}, B_m = \begin{bmatrix} \beta_\nu & \cdots & \beta_k & & & \\ \vdots & \ddots & & \ddots & & \\ \beta_0 & & \ddots & & \ddots & \\ & \ddots & & \ddots & & \beta_k \\ & & \ddots & & \ddots & \vdots \\ & & & \beta_0 & \cdots & \beta_\nu \end{bmatrix}_{(m-\nu)\times(m-\nu)}$$

and the vectors

$$\mathbf{u} = (u_\nu, \cdots, u_{m-1})^T, \quad \mathbf{f} = (f_\nu, \cdots, f_{m-1})^T.$$

Linear Multistep Methods in Boundary Value Form

If we collect the matrices for the inner steps of a *scalar* ODE, we get A_m, B_m, and the vectors

$$\mathbf{u} = (u_\nu, \cdots, u_{m-1})^T, \quad \mathbf{f} = (f_\nu, \cdots, f_{m-1})^T.$$

Finding the system

$$A_m\mathbf{u} - hB_m\mathbf{f} = - \begin{bmatrix} \sum_{j=0}^{\nu-1}(\alpha_j y_j - h\beta_j f_j) \\ \vdots \\ a_0 y_{\nu-1} - h\beta_0 f_{\nu-1} \\ 0 \\ \vdots \\ 0 \\ \alpha_k y_m - h\beta_k f_m \\ \vdots \\ \sum_{j=1}^{\mu}(\alpha_{\nu+j} y_{m-1+j} - h\beta_{\nu_1+j} f_{m-1+j}). \end{bmatrix}$$

- A_m and B_m are Toeplitz matrices with *lower bandwidth* ν and *upper bandwidth* μ.
- We still need **auxiliary formulas** to fix the $\nu + \mu - 1$ starting/ending values.

Convergence and stability

Before concluding the construction, let's focus on *convergence* and *stability*.

Convergence and stability

Before concluding the construction, let's focus on *convergence* and *stability*.

❓ Did we gain anything by moving on to a more difficult problem?

Convergence and stability

Before concluding the construction, let's focus on *convergence* and *stability*.

❓ Did we gain anything by moving on to a more difficult problem?

$S_{\nu,\mu}$-polynomial (Brugnano and Trigiante 1998, Definition 4.4.2)

A polynomial $p(z)$ of degree $k = \nu + \mu$ is an $S_{\nu,\mu}$-polynomial if its roots are such that

$$|z_1| \leq |z_2| \leq \cdots \leq |z_\nu| < 1 < |z_{\nu+1}| \leq \cdots \leq |z_\nu|.$$

Convergence and stability

Before concluding the construction, let's focus on *convergence* and *stability*.

❓ Did we gain anything by moving on to a more difficult problem?

$S_{\nu,\mu}$-polynomial (Brugnano and Trigiante 1998, Definition 4.4.2)

A polynomial $p(z)$ of degree $k = \nu + \mu$ is an $S_{\nu,\mu}$-polynomial if its roots are such that

$$|z_1| \leq |z_2| \leq \cdots \leq |z_\nu| < 1 < |z_{\nu+1}| \leq \cdots \leq |z_\nu|.$$

$N_{\nu,\mu}$-polynomial (Brugnano and Trigiante 1998, Definition 4.4.3)

A polynomial $p(z)$ of degree $k = \nu + \mu$ is an $S_{\nu,\mu}$-polynomial if its roots are such that

$$|z_1| \leq |z_2| \leq \cdots \leq |z_\nu| \leq 1 < |z_{\nu+1}| \leq \cdots \leq |z_\nu|.$$

being **simple** the roots of unit modulus.

Convergence and stability

Before concluding the construction, let's focus on *convergence* and *stability*.

❓ Did we gain anything by moving on to a more difficult problem?

$S_{\nu,\mu}$-polynomial (Brugnano and Trigiante 1998, Definition 4.4.2)

A polynomial $p(z)$ of degree $k = \nu + \mu$ is an $S_{\nu,\mu}$-polynomial if its roots are such that

$$|z_1| \leq |z_2| \leq \cdots \leq |z_\nu| < 1 < |z_{\nu+1}| \leq \cdots \leq |z_\nu|.$$

$N_{\nu,\mu}$-polynomial (Brugnano and Trigiante 1998, Definition 4.4.3)

A polynomial $p(z)$ of degree $k = \nu + \mu$ is an $S_{\nu,\mu}$-polynomial if its roots are such that

$$|z_1| \leq |z_2| \leq \cdots \leq |z_\nu| \leq 1 < |z_{\nu+1}| \leq \cdots \leq |z_\nu|.$$

being **simple** the roots of unit modulus.

If $\nu = k$ $(\mu = 0)$, these are the conditions for LMF 0-stability!

Convergence and stability

Let $a_{-\nu}a_{\mu} \neq 0$ and

$$T_n = \begin{bmatrix} a_0 & \cdots & a_\mu & & & \\ \vdots & \ddots & & \ddots & & \\ a_{-\nu} & & \ddots & & \ddots & \\ & \ddots & & \ddots & & a_\mu \\ & & \ddots & & \ddots & \vdots \\ & & & a_{-\nu} & \cdots & a_0 \end{bmatrix},$$

we consider the polynomial

$$p(z) = \sum_{i=-\nu}^{\mu} a_i z^{\nu+i}.$$

Convergence and stability

Let $a_{-\nu}a_\mu \neq 0$ and

$$T_n = \begin{bmatrix} a_0 & \cdots & a_\mu & & & \\ \vdots & \ddots & & \ddots & & \\ a_{-\nu} & & \ddots & & \ddots & \\ & \ddots & & \ddots & & a_\mu \\ & & \ddots & & \ddots & \vdots \\ & & & a_{-\nu} & \cdots & a_0 \end{bmatrix},$$

we consider the polynomial

$$p(z) = \sum_{i=-\nu}^{\mu} a_i z^{\nu+i}.$$

Lemma (Brugnano and Trigiante 1998, Lemma 4.4.4)

If the polynomial $p(z)$ associated with the matrix T_n is an $N_{\nu,\mu}$-polynomial, then T_n^{-1} has entries $t_{i,j}^{(-1)}$ such that

1. $|t_{i,j}^{(-1)}| \leq \gamma$ independent of N, for $i \geq j$,
2. $|t_{i,j}^{(-1)}| \leq \eta\xi^{j-i}$ for $i < j$, where $\eta > 0$ and $0 < \xi < 1$ are independent of N.

Convergence and stability

Let $a_{-\nu}a_{\mu} \neq 0$ and

$$T_n = \begin{bmatrix} a_0 & \cdots & a_\mu & & & \\ \vdots & \ddots & & \ddots & & \\ a_{-\nu} & & \ddots & & \ddots & \\ & \ddots & & \ddots & & a_\mu \\ & & \ddots & & \ddots & \vdots \\ & & & a_{-\nu} & \cdots & a_0 \end{bmatrix},$$

we consider the polynomial

$$p(z) = \sum_{i=-\nu}^{\mu} a_i z^{\nu+i}.$$

Lemma (Brugnano and Trigiante 1998, Lemma 4.4.4)

If the polynomial $p(z)$ associated with the matrix T_n is an $N_{\nu,\mu}$-polynomial, then T_n^{-1} has entries $t_{i,j}^{(-1)}$ such that

1. $|t_{i,j}^{(-1)}| \leq \gamma$ independent of N, for $i \geq j$,
2. $|t_{i,j}^{(-1)}| \leq \eta\xi^{j-i}$ for $i < j$, where $\eta > 0$ and $0 < \xi < 1$ are independent of N.

$$\|T_n^{-1}\| \leq \gamma C_n + \eta\Delta_n,$$

with $C_n = \begin{bmatrix} 1 & & \\ \vdots & \ddots & \\ 1 & \cdots & 1 \end{bmatrix}$,

Convergence and stability

Let $a_{-\nu}a_{\mu} \neq 0$ and

$$T_n = \begin{bmatrix} a_0 & \cdots & a_\mu & & & \\ \vdots & \ddots & & \ddots & & \\ a_{-\nu} & & \ddots & & \ddots & \\ & \ddots & & \ddots & & a_\mu \\ & & \ddots & & \ddots & \vdots \\ & & & a_{-\nu} & \cdots & a_0 \end{bmatrix},$$

we consider the polynomial

$$p(z) = \sum_{i=-\nu}^{\mu} a_i z^{\nu+i}.$$

Lemma (Brugnano and Trigiante 1998, Lemma 4.4.4)

If the polynomial $p(z)$ associated with the matrix T_n is an $N_{\nu,\mu}$-polynomial, then T_n^{-1} has entries $t_{i,j}^{(-1)}$ such that

1. $|t_{i,j}^{(-1)}| \leq \gamma$ independent of N, for $i \geq j$,
2. $|t_{i,j}^{(-1)}| \leq \eta \xi^{j-i}$ for $i < j$, where $\eta > 0$ and $0 < \xi < 1$ are independent of N.

👁 $||T_n^{-1}|| \leq \gamma C_n + \eta \Delta_n,$

with Δ_n the upper triangular Toeplitz matrix with last column $(\xi^{n-1}, \ldots, \xi^2, \xi, 0)^T$.

Convergence and stability

Theorem (Brugnano and Trigiante 1998, Theorem 4.4.3)

Ignoring the effect of round-off errors, a BVM with (ν, μ)-boundary conditions is convergent if it is consistent and the polynomial $\rho(z)$ is an $N_{\nu,\mu}$-polynomial.

Convergence and stability

Theorem (Brugnano and Trigiante 1998, Theorem 4.4.3)

Ignoring the effect of round-off errors, a BVM with (ν, μ)-boundary conditions is convergent if it is consistent and the polynomial $\rho(z)$ is an $N_{\nu,\mu}$-polynomial.

To reproduce the "0-stable + consistent $\Rightarrow$ convergence" framework, we define:

$0_{\nu,\mu}$-stability (Brugnano and Trigiante 1998, Definition 4.5.1)

A BVM with (ν, μ)-boundary conditions is $0_{\nu,\mu}$-stable if the corresponding polynomial $\rho(z)$ is an $N_{\nu,\mu}$-polynomial.

Convergence and stability

Theorem (Brugnano and Trigiante 1998, Theorem 4.4.3)

Ignoring the effect of round-off errors, a BVM with (ν, μ)-boundary conditions is convergent if it is consistent and the polynomial $\rho(z)$ is an $N_{\nu,\mu}$-polynomial.

To reproduce the "0-stable + consistent $\Rightarrow$ convergence" framework, we define:

$0_{\nu,\mu}$-stability (Brugnano and Trigiante 1998, Definition 4.5.1)

A BVM with (ν, μ)-boundary conditions is $0_{\nu,\mu}$-stable if the corresponding polynomial $\rho(z)$ is an $N_{\nu,\mu}$-polynomial.

(ν, μ)-Absolute stability (Brugnano and Trigiante 1998, Definition 4.7.1)

A BVM with (ν, μ)-boundary conditions is ν, μ-Absolutely stable for a given complex number q it the polynomial $\pi(z, q) = \rho(z) - q\sigma(z)$, is an $S_{\nu,\mu}$-polynomial.

Convergence and stability

We have a degree of **arbitrariness** in deciding how and how many initial / final conditions to set. Clearly ν has to be at least one (we do have an initial condition of our IVP), then for the remaining we have to let (ν, μ)-Absolute stability guide us.

Convergence and stability

We have a degree of **arbitrariness** in deciding how and how many initial / final conditions to set. Clearly ν has to be at least one (we do have an initial condition of our IVP), then for the remaining we have to let (ν, μ)-Absolute stability guide us.

Correct use
: a consistent LMF is *correctly used* in $q \in \mathbb{C}^{-}$, where $\pi(z, q)$ is an $S_{\nu,\mu}$-polynomial, if ν conditions are imposed at the initial points, and μ conditions are posed at the end of the interval.

Convergence and stability

We have a degree of **arbitrariness** in deciding how and how many initial / final conditions to set. Clearly ν has to be at least one (we do have an initial condition of our IVP), then for the remaining we have to let (ν, μ)-Absolute stability guide us.

Correct use
: a consistent LMF is *correctly used* in $q \in \mathbb{C}^-$, where $\pi(z, q)$ is an $S_{\nu,\mu}$-polynomial, if ν conditions are imposed at the initial points, and μ conditions are posed at the end of the interval.

To have a livable life, one always consider family of methods for which the boundary of the (ν, μ)-Absolutely stability region is a *regular Jordan curve*. More specifically, having that

$$\mathcal{A}_{\nu,\mu} = \{q \in \mathbb{C} \,:\, \pi(z, q) \text{ is an } S_{\nu,\mu}\text{-polynomial}\},$$

has the origin on its boundary and is possibly equal to the whole $\mathbb{C}^-$.

A gallery of formulas

It is possible to reformulate many LMFs in this new format

A gallery of formulas

It is possible to reformulate many LMFs in this new format

- BDF $\Rightarrow$ Generalized-BDF (GBDF): $\sum_{i=0}^{k} \alpha_i u_{n+i} = h f_{n+j}$, $j \in \{0, 1, \ldots, k\}$

A gallery of formulas

It is possible to reformulate many LMFs in this new format

- BDF $\Rightarrow$ Generalized-BDF (GBDF): $\sum_{i=0}^{k} \alpha_i u_{n+i} = h f_{n+j}$, $j \in \{0, 1, \ldots, k\}$
  - A method of this form is $0_{\nu,k-\nu}$-stable and $A_{\nu,k-\nu}$-stable for

$$\nu = \begin{cases} \frac{k+2}{2}, & \text{for even } k, \\ \frac{k+1}{2}, & \text{for odd } k. \end{cases}$$

  $\Rightarrow$ with this choice we no longer have the constraint of having at most $k = 6$ steps of the standard BDF!

- Adams-Moulton Methods $\Rightarrow$ GAMM $u_{n+j} - u_{n+j-1} = h \sum_{i=0}^{k} \beta_i f_{n+i}$

A gallery of formulas

It is possible to reformulate many LMFs in this new format

- BDF $\Rightarrow$ Generalized-BDF (GBDF): $\sum_{i=0}^{k} \alpha_i u_{n+i} = h f_{n+j}$, $j \in \{0, 1, \dots, k\}$
  - A method of this form is $0_{\nu,k-\nu}$-stable and $A_{\nu,k-\nu}$-stable for

$$\nu = \begin{cases} \frac{k+2}{2}, & \text{for even } k, \\ \frac{k+1}{2}, & \text{for odd } k. \end{cases}$$

$\Rightarrow$ with this choice we no longer have the constraint of having at most $k = 6$ steps of the standard BDF!

- Adams-Moulton Methods $\Rightarrow$ GAMM $u_{n+j} - u_{n+j-1} = h \sum_{i=0}^{k} \beta_i f_{n+i}$
  - A method of this form is $0_{\nu,k-\nu}$-stable and $A_{\nu,k-\nu}$-stable for

$$\nu = \begin{cases} \frac{k+1}{2}, & \text{for odd } k, \\ \frac{k}{2}, & \text{for even } k. \end{cases}$$

See the book (Brugnano and Trigiante 1998) for other possible generalizations.

Additional formulas

We need **additional formulas** for the $k-1 = \nu + \mu - 1$ boundary values.

Additional formulas

We need **additional formulas** for the $k-1 = \nu + \mu - 1$ boundary values.

$$A_m = \begin{bmatrix} 1 & \dots & 0 & & & \\ \alpha_0^{(1)} & \dots & \alpha_k^{(1)}, & & & \\ \vdots & & \vdots & & & \\ \alpha_0^{(\nu-1)} & \dots & \alpha_k^{(\nu-1)} & & & \\ \alpha_0 & \dots & \alpha_k & & & \\ & \alpha_0 & \dots & \alpha_k & & \\ & & \ddots & \ddots & \ddots & \\ & & & \alpha_0 & \dots & \alpha_k \\ & & & \alpha_0^{(m-k+\nu+1)} & \dots & \alpha_k^{(m-k+\nu+1)} \\ & & & \vdots & & \vdots \\ & & & \alpha_0^{(m)} & \dots & \alpha_k^{(m)} \end{bmatrix},$$

Additional formulas

We need **additional formulas** for the $k-1=\nu+\mu-1$ boundary values.

$$
B_m = \begin{bmatrix}
0 & \dots & 0 & & & \\
\beta_0^{(1)} & \dots & \beta_k^{(1)}, & & & \\
\vdots & & \vdots & & & \\
\beta_0^{(\nu-1)} & \dots & \beta_k^{(\nu-1)} & & & \\
\beta_0 & \dots & \beta_k & & & \\
& \beta_0 & \dots & \beta_k & & \\
& & \ddots & \ddots & \ddots & \\
& & & \beta_0 & \dots & \beta_k \\
& & & \beta_0^{(m-k+\nu+1)} & \dots & \beta_k^{(m-k+\nu+1)}, \\
& & & \vdots & & \vdots \\
& & & \beta_0^{(m)} & \dots & \beta_k^{(m)}
\end{bmatrix}.
$$

Additional formulas

We need **additional formulas** for the $k-1 = \nu + \mu - 1$ boundary values.

- If we know how to compute them, then we end up having to solve the **matrix equation**

$$M_n U A_m^T - h L_n U B_m^T = F,$$

or the **linear system**

$$(A_m \otimes M_n - h B_m \otimes L_n)\mathbf{u} = \mathbf{f}, \text{ where } \mathrm{vec}(U) = \mathbf{u}, \ \mathrm{vec}(F) = \mathbf{f}.$$

Additional formulas

- We need **additional formulas** for the $k-1=\nu+\mu-1$ boundary values.
  - If we know how to compute them, then we end up having to solve the **matrix equation**
    $$M_n U A_m^T - h L_n U B_m^T = F,$$
    or the **linear system**
    $$(A_m \otimes M_n - h B_m \otimes L_n)\mathbf{u} = \mathbf{f}, \text{ where } \operatorname{vec}(U) = \mathbf{u}, \operatorname{vec}(F) = \mathbf{f}.$$
- Let us build everything for using GBDFs and our fractional-in-space problem.

Generalized BDF

First we need to compute $\rho(z)$ and $\sigma(z)$

```
function [ro,si] = rosi_bdf( k, j )
b  = zeros(k+1,1); b(2) = 1;
ro = vsolve( -j:k-j, b(:) );
si = zeros( k+1, 1 ); si( j+1 ) = 1;
end
```

Coefficients are computed by **imposing consistency** of maximal order p:

$$\sum_{j=0}^{k} \left(j^s \alpha_j - s j^{s-1} \beta_j \right) = 0,$$

$s = 0, 1, \ldots, p.$

Generalized BDF

First we need to compute $\rho(z)$ and $\sigma(z)$

```
function [ro,si] = rosi_bdf( k, j )
b  = zeros(k+1,1); b(2) = 1;
ro = vsolve( -j:k-j, b(:) );
si = zeros( k+1, 1 ); si( j+1 ) = 1;
end
```

Coefficients are computed by **imposing consistency** of maximal order p:

$$\sum_{j=0}^{k} (j^s \alpha_j - s j^{s-1} \beta_j) = 0,$$

$s = 0, 1, \ldots, p.$

```
function f = vsolve( x, b )
f = b;
n = length( x )-1;
for k = 1:n
 for i = n+1:-1:k+1
  f(i) = f(i) - x(k)*f(i-1);
 end
end
for k = n:-1:1
 for i = k+1:n+1
  f(i) = f(i)/( x(i) - x(i-k) );
 end
 for i = k:n
  f(i) = f(i) - f(i+1);
 end
end
end
```

Generalized BDF

Then we use the `ro_si` routine to build the A_m and B_m matrices

```
function [a,b] = mab( k, n )
nu = fix( (k+2)/2 );
a  = spalloc( n, n+1, (k+1)*n );
b = a;
for i = 1:nu
 [ro,si] = rosi_bdf( k, i );
 a(i,1:k+1) = ro.';
 b(i,1:k+1) = si.';
end
for i = nu+1:n-(k-nu)
 a(i,i+1+(-nu:k-nu)) = ro.';
 b(i,i+1+(-nu:k-nu)) = si.';
end
```

```
j = nu;
for i = n-(k-nu)+1:n
 j = j + 1;
 [ro,si] = rosi_bdf( k, j );
 a(i,n+1+(-k:0)) = ro.';
 b(i,n+1+(-k:0)) = si.';
end
end
```

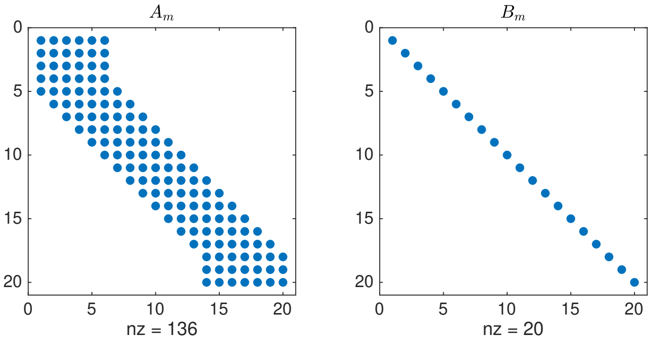
- `for i = 1:nu; end`, initial conditions,
- `for i = nu+1:n-(k-nu); end`, Toepltiz part,
- `for i = n-(k-nu)+1:n; end`, final conditions.

Generalized BDF

We can use the routine to generate

```
[Alpha,Beta] = mab(k,m); A = Alpha(:,2:m+1); B = Beta(:,2:m+1);
```

and visualize them



👁 The first column contains the coefficients needed to compute the right-hand-side.

Generalized BDF

We now need to build the right-hand-side

```
nk=n*(m+1);
b=zeros(nk,1);  % Allocate the space for one more than needed
for j=1:m  % Use the source to build the rhs:
 b(1+j*n:(j+1)*n)=f(x,t0+j*h);
end
b(n+1:n*(m+1))=h*kron(Beta,speye(n))*b; % Correct with the betas coeff.s
b(1:n)=u0;                       % First block as the initial condition
% Correction coefficients:
Am = kron(Alpha(:,1),speye(n))-h*kron(Beta(:,1),L);
b(n+1:nk)=b(n+1:nk)-Am*u0; % Finish building RHS
```

And then we can solve the linear system

```
Mat = kron(A,M) - h*kron(B,L); rhs = b(n+1:nk);
u = Mat\rhs;
```

Generalized BDF

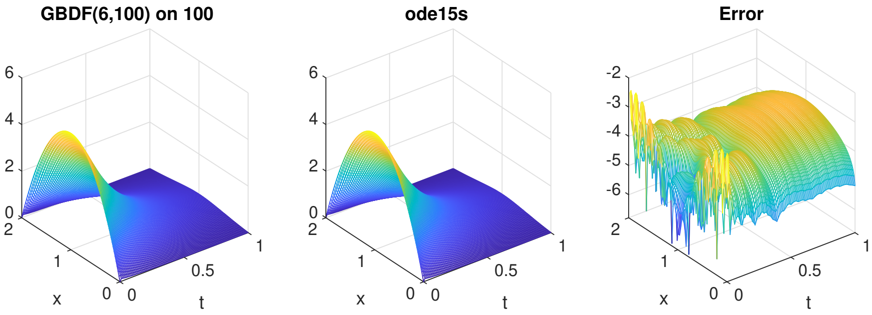
We can compare the solution with ode15s:

```
U = [u0,reshape(u,n,m)]; t = t0:h:tf;
[TT,UU] = ode15s(@(t,y) L*y +
↪  f(x.',t),t,u0);
E = abs(U-reshape(UU,m+1,n).');
figure(2)
subplot(1,3,1)
mesh(t,x,U);
xlabel('t');
ylabel('x');
title('GBDF(6,100) on 100')
```

```
subplot(1,3,2)
mesh(t,x,reshape(UU,m+1,n).')
xlabel('t');
ylabel('x');
title('ode15s')
subplot(1,3,3)
mesh(t,x,log10())
xlabel('t');
ylabel('x');
title('Error')
```

Generalized BDF

We can compare the solution with `ode15s`:



❓ What happens if we attempt solution via our matrix-equation solver?

Generalized BDF

We can solve it by doing:

```
maxit = 100;
tol = 1e-9;
[LL,UL] = lu(-h*L);
[LA,UA] = lu(A);
[X1,X2,res]=kpik_sylv(-h*L,LL,UL,A,
↪  LA,UA,C1,C2,maxit,tol);
```

Using our non-symmetric test problem with variable coefficients and fractional order α.

Generalized BDF

We can solve it by doing:

```
maxit = 100;
tol = 1e-9;
[LL,UL] = lu(-h*L);
[LA,UA] = lu(A);
[X1,X2,res]=kpik_sylv(-h*L,LL,UL,A,
↪  LA,UA,C1,C2,maxit,tol);
```

Using our non-symmetric test problem with variable coefficients and fractional order α.

k	m	n	IT	Res.
2	32	64	16	1.08e-15
2	64	128	23	2.16e-10
2	128	256	30	4.72e-10
2	256	512	38	9.20e-10
2	512	1024	49	7.31e-10
2	1024	2048	62	7.82e-10
2	2048	4096	78	8.06e-10
2	4096	8192	97	9.24e-10

Generalized BDF

We can solve it by doing:

```
maxit = 100;
tol = 1e-9;
[LL,UL] = lu(-h*L);
[LA,UA] = lu(A);
[X1,X2,res]=kpik_sylv(-h*L,LL,UL,A,
↪  LA,UA,C1,C2,maxit,tol);
```

Using our non-symmetric test problem with variable coefficients and fractional order α.

k	m	n	IT	Res.
3	32	64	15	7.18e-10
3	64	128	20	9.80e-10
3	128	256	26	7.77e-10
3	256	512	34	4.21e-10
3	512	1024	43	5.75e-10
3	1024	2048	54	8.05e-10
3	2048	4096	68	8.84e-10
3	4096	8192	85	9.87e-10

Generalized BDF

We can solve it by doing:

```
maxit = 100;
tol = 1e-9;
[LL,UL] = lu(-h*L);
[LA,UA] = lu(A);
[X1,X2,res]=kpik_sylv(-h*L,LL,UL,A,
↪  LA,UA,C1,C2,maxit,tol);
```

Using our non-symmetric test problem with variable coefficients and fractional order α.

k	m	n	IT	Res.
4	32	64	16	1.19e-14
4	64	128	24	3.22e-10
4	128	256	31	4.05e-10
4	256	512	39	6.97e-10
4	512	1024	50	6.20e-10
4	1024	2048	63	7.70e-10
4	2048	4096	79	9.05e-10
4	4096	8192	99	9.05e-10

Generalized BDF

We can solve it by doing:

```
maxit = 100;
tol = 1e-9;
[LL,UL] = lu(-h*L);
[LA,UA] = lu(A);
[X1,X2,res]=kpik_sylv(-h*L,LL,UL,A,
↪  LA,UA,C1,C2,maxit,tol);
```

Using our non-symmetric test problem with variable coefficients and fractional order α.

k	m	n	IT	Res.
5	32	64	16	1.72e-14
5	64	128	22	2.96e-10
5	128	256	28	4.90e-10
5	256	512	36	5.56e-10
5	512	1024	46	5.53e-10
5	1024	2048	58	7.10e-10
5	2048	4096	73	8.04e-10
5	4096	8192	91	9.75e-10

Generalized BDF

We can solve it by doing:

```
maxit = 100;
tol = 1e-9;
[LL,UL] = lu(-h*L);
[LA,UA] = lu(A);
[X1,X2,res]=kpik_sylv(-h*L,LL,UL,A,
↪ LA,UA,C1,C2,maxit,tol);
```

Using our non-symmetric test problem with variable coefficients and fractional order α.

k	m	n	IT	Res.
6	32	64	16	3.46e-14
6	64	128	24	4.70e-10
6	128	256	31	5.73e-10
6	256	512	40	4.78e-10
6	512	1024	50	9.39e-10
6	1024	2048	64	7.69e-10
6	2048	4096	81	7.31e-10
6	4096	8192	100	1.10e-09

Generalized BDF

We can solve it by doing:

```
maxit = 100;
tol = 1e-9;
[LL,UL] = lu(-h*L);
[LA,UA] = lu(A);
[X1,X2,res]=kpik_sylv(-h*L,LL,UL,A,
↪ LA,UA,C1,C2,maxit,tol);
```

Using our non-symmetric test problem with variable coefficients and fractional order α.

k	m	n	IT	Res.
7	32	64	16	6.13e-15
7	64	128	22	6.60e-10
7	128	256	29	4.78e-10
7	256	512	37	7.04e-10
7	512	1024	47	8.47e-10
7	1024	2048	60	7.66e-10
7	2048	4096	76	7.36e-10
7	4096	8192	95	8.46e-10

Generalized BDF

We can solve it by doing:

```
maxit = 100;
tol = 1e-9;
[LL,UL] = lu(-h*L);
[LA,UA] = lu(A);
[X1,X2,res]=kpik_sylv(-h*L,LL,UL,A,
↪  LA,UA,C1,C2,maxit,tol);
```

Using our non-symmetric test problem with variable coefficients and fractional order α.

k	m	n	IT	Res.
8	32	64	16	2.46e-14
8	64	128	24	5.41e-10
8	128	256	31	7.57e-10
8	256	512	40	6.53e-10
8	512	1024	51	7.34e-10
8	1024	2048	65	6.98e-10
8	2048	4096	82	7.42e-10
8	4096	8192	100	1.56e-09

Generalized BDF

We can solve it by doing:

```
maxit = 100;
tol = 1e-9;
[LL,UL] = lu(-h*L);
[LA,UA] = lu(A);
[X1,X2,res]=kpik_sylv(-h*L,LL,UL,A,
↪  LA,UA,C1,C2,maxit,tol);
```

Using our non-symmetric test problem with variable coefficients and fractional order α.

k	m	n	IT	Res.
8	32	64	16	2.46e-14
8	64	128	24	5.41e-10
8	128	256	31	7.57e-10
8	256	512	40	6.53e-10
8	512	1024	51	7.34e-10
8	1024	2048	65	6.98e-10
8	2048	4096	82	7.42e-10
8	4096	8192	100	1.56e-09

- The solution seems to be robust with respect to k,

Generalized BDF

We can solve it by doing:

```
maxit = 100;
tol = 1e-9;
[LL,UL] = lu(-h*L);
[LA,UA] = lu(A);
[X1,X2,res]=kpik_sylv(-h*L,LL,UL,A,
↪  LA,UA,C1,C2,maxit,tol);
```

Using our non-symmetric test problem with variable coefficients and fractional order α.

k	m	n	IT	Res.
8	32	64	16	2.46e-14
8	64	128	24	5.41e-10
8	128	256	31	7.57e-10
8	256	512	40	6.53e-10
8	512	1024	51	7.34e-10
8	1024	2048	65	6.98e-10
8	2048	4096	82	7.42e-10
8	4096	8192	100	1.56e-09

- The solution seems to be robust with respect to k,
- We still have a small increase with n and m.

Structured preconditioner

Let's now look for a different approach.

Structured preconditioner

Let's now look for a different approach.

- We can do matrix vector products with the system matrix without assembling the matrix:

```
function [y] = Mprod(A,B,L,h,x)
[sp1,~] = size(A);
[m,~] = size(L);
X = reshape(x,m,sp1);
Y = X*A' - h*(L*X*B');
y = reshape(Y,m*sp1,1);
end
```

Structured preconditioner

Let's now look for a different approach.

- We can do matrix vector products with the system matrix without assembling the matrix:

```
function [y] = Mprod(A,B,L,h,x)
[sp1,~] = size(A);
[m,~] = size(L);
X = reshape(x,m,sp1);
Y = X*A' - h*(L*X*B');
y = reshape(Y,m*sp1,1);
end
```

- The linear system is **not symmetric**: we can use either GMRES or Flexible-GMRES to solve it.

Structured preconditioner

Let's now look for a different approach.

- We can do matrix vector products with the system matrix without assembling the matrix:

```
function [y] = Mprod(A,B,L,h,x)
[sp1,~] = size(A);
[m,~] = size(L);
X = reshape(x,m,sp1);
Y = X*A' - h*(L*X*B');
y = reshape(Y,m*sp1,1);
end
```

- The linear system is **not symmetric**: we can use either GMRES or Flexible-GMRES to solve it.
- We just need to *figure out* a *preconditioner*.

Structured preconditioner

The idea is *again* using a preconditioner that has the same structure:

$$P = \breve{A}_m \otimes M_n - h\breve{B}_m \otimes \tilde{L}_n,$$

This idea comes from (Bertaccini 2000, 2001; Bertaccini and Ng 2001),

Structured preconditioner

The idea is *again* using a preconditioner that has the same structure:

$$P = \breve{A}_m \otimes M_n - h\breve{B}_m \otimes \tilde{L}_n,$$

- This idea comes from (Bertaccini 2000, 2001; Bertaccini and Ng 2001),
- How do we select the approximations $\breve{A}_m$, $\breve{B}_m$ and $\tilde{L}_n$?

Structured preconditioner

The idea is *again* using a preconditioner that has the same structure:

$$P = \breve{A}_m \otimes M_n - h\breve{B}_m \otimes \tilde{L}_n,$$

- This idea comes from (Bertaccini 2000, 2001; Bertaccini and Ng 2001),
- How do we select the approximations $\breve{A}_m$, $\breve{B}_m$ and $\tilde{L}_n$?
  - A_m, B_m are Toeplitz + low-rank $\Rightarrow$ Circulant or Fast-Transform preconditioners,
  - $\tilde{L}_n$ has the *quasi-Toeplitz structure* we have seen, so we can use some of the techniques we had already seen for this; (Bertaccini and Durastante 2018).

Structured preconditioner

The idea is *again* using a preconditioner that has the same structure:

$$P = \breve{A}_m \otimes M_n - h\breve{B}_m \otimes \tilde{L}_n,$$

- This idea comes from (Bertaccini 2000, 2001; Bertaccini and Ng 2001),
- How do we select the approximations $\breve{A}_m$, $\breve{B}_m$ and $\tilde{L}_n$?
  - A_m, B_m are Toeplitz $+$ low-rank $\Rightarrow$ Circulant or Fast-Transform preconditioners,
  - $\tilde{L}_n$ has the *quasi-Toeplitz structure* we have seen, so we can use some of the techniques we had already seen for this; (Bertaccini and Durastante 2018).
- It would be good to also have a **parallel way of applying the preconditioner**.

Structured preconditioner

If $\breve{A}_m$ and $\breve{B}_m$ are *circulant–like approximations* of the Toeplitz (+ "low rank") matrices A_m and B_m, and the mass matrix is the identity, then we can express the **eigenvalues** of P as

$$\phi_i - h\psi_i\lambda_j, \qquad i = 1, \ldots, m, \quad j = 1, \ldots, n,$$

where

- $\{\phi_i\}$ are the eigenvalues of the circulant–like approximation $\breve{A}$,
- $\{\psi_i\}$ are the eigenvalues of the circulant–like approximation $\breve{B}$,
- $\{\lambda_j\}$ are the eigenvalues of the selected approximation of J_n.

Structured preconditioner

- If $\breve{A}_m$ and $\breve{B}_m$ are *circulant–like approximations* of the Toeplitz ($+$ "low rank") matrices A_m and B_m, and the mass matrix is the identity, then we can express the **eigenvalues** of P as

$$\phi_i - h\psi_i\lambda_j, \qquad i = 1, \ldots, m, \quad j = 1, \ldots, n,$$

  where
  - $\{\phi_i\}$ are the eigenvalues of the circulant–like approximation $\breve{A}$,
  - $\{\psi_i\}$ are the eigenvalues of the circulant–like approximation $\breve{B}$,
  - $\{\lambda_j\}$ are the eigenvalues of the selected approximation of J_n.
- What circulant-like approximation do we want?

Structured preconditioner

- If $\breve{A}_m$ and $\breve{B}_m$ are *circulant–like approximations* of the Toeplitz (+ "low rank") matrices A_m and B_m, and the mass matrix is the identity, then we can express the **eigenvalues** of P as

$$\phi_i - h\psi_i\lambda_j, \qquad i = 1, \ldots, m, \quad j = 1, \ldots, n,$$

  where
  - $\{\phi_i\}$ are the eigenvalues of the circulant–like approximation $\breve{A}$,
  - $\{\psi_i\}$ are the eigenvalues of the circulant–like approximation $\breve{B}$,
  - $\{\lambda_j\}$ are the eigenvalues of the selected approximation of J_n.
- What circulant-like approximation do we want?
  - An idea could be using Strang approximation (Gu et al. 2015)

$$P_{\mathfrak{s}} = \mathfrak{s}(A_m) \otimes I_m - h\mathfrak{s}(B_m) \otimes L_n,$$

Structured preconditioner

$$P_{\mathfrak{s}} = \mathfrak{s}(A_m) \otimes I_m - h\mathfrak{s}(B_m) \otimes L_n,$$

$$\mathfrak{s}(A) = \begin{bmatrix}
\alpha_\nu & \cdots & \alpha_k & & & & \alpha_0 & \cdots & \alpha_{\nu-1} \\
\vdots & \ddots & & \ddots & & & & \ddots & \vdots \\
\alpha_0 & & \ddots & & \ddots & & & & \alpha_0 \\
 & \ddots & & \ddots & & \ddots & & 0 & \\
 & & \ddots & & \ddots & & \ddots & & \\
 & 0 & & \ddots & & \ddots & & \ddots & \\
\alpha_k & & & & \ddots & & \ddots & & \alpha_k \\
\vdots & \ddots & & & & \ddots & & \ddots & \vdots \\
\alpha_{\nu+1} & \cdots & \alpha_k & & & & \alpha_0 & \cdots & \alpha_\nu
\end{bmatrix},$$

- $\mathfrak{s}(B)$ can be built analogously.

Structured preconditioner

$$P_{\mathfrak{s}} = \mathfrak{s}(A_m) \otimes I_m - h\mathfrak{s}(B_m) \otimes L_n,$$

$$\mathfrak{s}(A) = \begin{bmatrix}
\alpha_\nu & \cdots & \alpha_k & & & & \alpha_0 & \cdots & \alpha_{\nu-1} \\
\vdots & \ddots & & \ddots & & & & \ddots & \vdots \\
\alpha_0 & & \ddots & & \ddots & & & & \alpha_0 \\
 & \ddots & & \ddots & & \ddots & & 0 & \\
 & & \ddots & & \ddots & & \ddots & & \\
 & 0 & & \ddots & & \ddots & & \ddots & \\
\alpha_k & & & & \ddots & & \ddots & & \alpha_k \\
\vdots & \ddots & & & & \ddots & & \ddots & \vdots \\
\alpha_{\nu+1} & \cdots & \alpha_k & & & & \alpha_0 & \cdots & \alpha_\nu
\end{bmatrix},$$

- $\mathfrak{s}(B)$ can be built analogously.
- $\mathfrak{s}(A)$ is singular due to the consistency condition.

Structured preconditioner

$$P_{\tilde{\mathfrak{s}}} = \tilde{\mathfrak{s}}(A)_m \otimes I_n - h\tilde{\mathfrak{s}}(B)_m \otimes L_n.$$

$$\mathfrak{s}(A) = \begin{bmatrix} \alpha_\nu & \cdots & \alpha_k & & & & \alpha_0 & \cdots & \alpha_{\nu-1} \\ \vdots & \ddots & & \ddots & & & & \ddots & \vdots \\ \alpha_0 & & \ddots & & \ddots & & & & \alpha_0 \\ & \ddots & & \ddots & & \ddots & & 0 & \\ & & \ddots & & \ddots & & \ddots & & \\ & 0 & & \ddots & & \ddots & & \ddots & \\ \alpha_k & & & & \ddots & & \ddots & & \alpha_k \\ \vdots & \ddots & & & & \ddots & & \ddots & \vdots \\ \alpha_{\nu+1} & \cdots & \alpha_k & & & & \alpha_0 & \cdots & \alpha_\nu \end{bmatrix},$$

- $\mathfrak{s}(B)$ can be built analogously.
- $\mathfrak{s}(A)$ is singular due to the consistency condition.
- It is a single 0 eigenvalue, so we can move it by a rank 1 perturbation: $\tilde{\mathfrak{s}}(\cdot)$.

Structured preconditioner

$$P_{\tilde{\mathfrak{s}}} = \tilde{\mathfrak{s}}(A)_m \otimes I_n - h\tilde{\mathfrak{s}}(B)_m \otimes L_n.$$

$$\mathfrak{s}(A) = \begin{bmatrix} \alpha_\nu & \cdots & \alpha_k & & & \alpha_0 & \cdots & \alpha_{\nu-1} \\ \vdots & \ddots & & \ddots & & & \ddots & \vdots \\ \alpha_0 & & \ddots & & \ddots & & & \alpha_0 \\ & \ddots & & \ddots & & \ddots & 0 & \\ & & \ddots & & \ddots & & \ddots & \\ & 0 & & \ddots & & \ddots & & \ddots \\ \alpha_k & & & & \ddots & & \ddots & \alpha_k \\ \vdots & \ddots & & & & \ddots & \ddots & \vdots \\ \alpha_{\nu+1} & \cdots & \alpha_k & & & \alpha_0 & \cdots & \alpha_\nu \end{bmatrix},$$

Proposition (Bertaccini 2001, Proposition 4.1)

If L has eigenvalues μ_r such that $\Re(\mu_r) < -\delta < 0$, $r = 1, \ldots, m$. Then the preconditioner $P_{\tilde{\mathfrak{s}}}$ is invertible for $A_{\nu,k-\nu}$-stable formulae.

Structured preconditioner

❓ What can we say about the **clustering properties** of this preconditioner?

Theorem (Bertaccini 2000, Theorem 4.1)

Let $\mathcal{M} = A_m \otimes I_n - hB_m \otimes L_n$ for an $A_{\nu,k-\nu}$-stable formulae with k steps. Let P be the block circulant preconditioner

$$P = \breve{A}_m \otimes M_n - h\breve{B}_m \otimes L_n.$$

Then, for fixed $\delta > 0$, there exists $C_\delta \geq 0$, $m_\delta \geq k$ such that, for all $m \geq m_\delta$ ($m+1$ is the size of A and B),

$$P^{-1}M = I + M_\delta^{(1)} + M_\delta^{(2)},$$

where $\mathrm{rank}(M_\delta^{(2)}) \leq n[2(k+1) + C_\delta]$ and $\|M_\delta^{(1)}\|_2 \leq \delta c_L$ does not depend on m. If P is defined as Strang's circulant preconditioner, then $C_\delta = \|M_\delta^{(1)}\| = 0$.

Structured preconditioner

Another available choice is using instead $\{\omega\}$-Circulant matrices, i.e.,

$$P_\omega = \omega(A_m) \otimes I_n - h\omega(B_m) \otimes L_n,$$

$$\omega(A_m) = \begin{bmatrix}
\alpha_\nu & \cdots & \alpha_k & & & & \omega\alpha_0 & \cdots & \omega\alpha_{\nu-1} \\
\vdots & \ddots & & \ddots & & & & \ddots & \vdots \\
\alpha_0 & & \ddots & & \ddots & & & & \omega\alpha_0 \\
 & \ddots & & \ddots & & \ddots & & 0 & \\
 & & \ddots & & \ddots & & \ddots & & \\
 & 0 & & \ddots & & \ddots & & \ddots & \\
\omega\alpha_k & & & & \ddots & & \ddots & & \alpha_k \\
\vdots & \ddots & & & & \ddots & & \ddots & \vdots \\
\omega\alpha_{\nu+1} & \cdots & \omega\alpha_k & & & & \alpha_0 & \cdots & \alpha_\nu
\end{bmatrix},$$

- $\omega(B_m)$ is defined similarly.
- The usual choice is setting $\omega = -1$, i.e., the skew-circulant preconditioner.

Stuctured preconditioner: application

To apply

$$P_\omega^{-1}\mathbf{v} = (\omega(A_m) \otimes I_n - h\omega(B_m) \otimes L_n)^{-1}\mathbf{v},$$

We can use the **diagonalization** of $\omega(A_m)$ and $\omega(B_m)$, i.e.,

$$P_\omega^{-1}\mathbf{v} = (F\Omega \otimes I_n)^{-1}(\Lambda_A \otimes I_n - h\Lambda_B \otimes L_n)^{-1}(\Omega^H F^H \otimes I_n)^{-1}\mathbf{v}.$$

1. Compute $\mathbf{w} = (\Omega^* F^* \otimes I_m)^{-1}\mathbf{v} = -V\Omega^{-H}F$,
2. Solve $(\Lambda_A \otimes I_n - h\Lambda_B \otimes L_n)^{-1}\mathbf{w}$ by solving

   $$(\lambda_i(A) I_n - h\lambda_i(B) L_n)\mathbf{z}_i = \mathbf{w}_i, \quad i = 1, \ldots, m$$

   with $\mathrm{vec}([\mathbf{w}_1, \ldots, \mathbf{w}_m]) = \mathbf{w}$, and similarly for $\mathbf{z}$,
3. Compute $\mathbf{y} = (F\Omega \otimes I_n)^{-1}\mathbf{z} = -ZF^H\Omega^{-1}$.

Stuctured preconditioner: application

To apply

$$P_\omega^{-1}\mathbf{v} = (\omega(A_m) \otimes I_n - h\omega(B_m) \otimes L_n)^{-1}\mathbf{v},$$

We can use the **diagonalization** of $\omega(A_m)$ and $\omega(B_m)$, i.e.,

$$P_\omega^{-1}\mathbf{v} = (F\Omega \otimes I_n)^{-1}(\Lambda_A \otimes I_n - h\Lambda_B \otimes L_n)^{-1}(\Omega^H F^H \otimes I_n)^{-1}\mathbf{v}.$$

1. Compute $\mathbf{w} = (\Omega^* F^* \otimes I_m)^{-1}\mathbf{v} = -V\Omega^{-H}F$,
2. Solve $(\Lambda_A \otimes I_n - h\Lambda_B \otimes L_n)^{-1}\mathbf{w}$ by solving

$$(\lambda_i(A) I_n - h\lambda_i(B) L_n)\mathbf{z}_i = \mathbf{w}_i, \quad i = 1, \ldots, m$$

   with $\mathrm{vec}([\mathbf{w}_1, \ldots, \mathbf{w}_m]) = \mathbf{w}$, and similarly for $\mathbf{z}$,
3. Compute $\mathbf{y} = (F\Omega \otimes I_n)^{-1}\mathbf{z} = -ZF^H\Omega^{-1}$.

! This step is **embarrassingly parallel**!

Numerical example

We use our favorite test problem with the space variant, nonsymmetric fractional operator in space and $\alpha = 1.5$, using GMRES(20) with a tolerance of `1e-9` using the P_{-1} preconditioner.

$k=2$			$k=3$			$k=4$			$k=5$			$k=6$		
n	m	It	n	m	It	n	m	It	n	m	It	n	m	It
64	32	30	64	32	32	64	32	35	64	32	38	64	32	46
128	64	31	128	64	33	128	64	38	128	64	45	128	64	53
256	128	31	256	128	34	256	128	39	256	128	48	256	128	58
512	256	31	512	256	34	512	256	39	512	256	50	512	256	62
1024	512	30	1024	512	33	1024	512	37	1024	512	49	1024	512	60

Numerical example

We use our favorite test problem with the space variant, nonsymmetric fractional operator in space and $\alpha = 1.5$, using GMRES(20) with a tolerance of `1e-9` using the P_{-1} preconditioner.

$k=2$		
n	m	It
64	32	30
128	64	31
256	128	31
512	256	31
1024	512	30

$k=3$		
n	m	It
64	32	32
128	64	33
256	128	34
512	256	34
1024	512	33

$k=4$		
n	m	It
64	32	35
128	64	38
256	128	39
512	256	39
1024	512	37

$k=5$		
n	m	It
64	32	38
128	64	45
256	128	48
512	256	50
1024	512	49

$k=6$		
n	m	It
64	32	46
128	64	53
256	128	58
512	256	62
1024	512	60

Reduced **iteration dependence**, but paid with **full memory price**!

Further modifications

We can further approximate the preconditioner by selecting instead of L_n in

$$P_\omega^{-1}\mathbf{v} = (\omega(A_m) \otimes I_n - h\omega(B_m) \otimes L_n)^{-1}\mathbf{v},$$

a suitable approximation, e.g.,

- $g_k(L_n)$ a bandwidth k approximation of the dense L_n matrix, i.e., using the information on the decay of the coefficients (Bertaccini and Durastante 2018).
- A *structured preconditioner* based on GLT theory.

Further modifications

We can further approximate the preconditioner by selecting instead of L_n in

$$P_{\omega}^{-1}\mathbf{v} = (\omega(A_m) \otimes I_n - h\omega(B_m) \otimes L_n)^{-1}\mathbf{v},$$

a suitable approximation, e.g.,

- $g_k(L_n)$ a bandwidth k approximation of the dense L_n matrix, i.e., using the information on the decay of the coefficients (Bertaccini and Durastante 2018).
- A *structured preconditioner* based on GLT theory.

Open areas of research

- Efficient solution strategies for the $\lambda_i(A)I_n - h\lambda_i(B)L_n$ systems,
- Load-balancing issues for parallelism,
- Optimal poles selection for the matrix-equation based solvers,
- Multigrid solvers/preconditioners for $(A_m \otimes M_n - hB_m \otimes L_n)\mathbf{u} = \mathbf{f}$.

Tensor Equations

A *different approach* that can be of interest is to use another structure.

Tensor Equations

A *different approach* that can be of interest is to use another structure.

Let us suppose that L_n is obtained as the discretization of a *multidimensional fractional operator*, i.e.,

$$L_n = \sum_{i=1}^{\ell} \left(K^{-}_{m,\ell} \bigotimes_{p=1}^{i-1} I \otimes G^{(\ell)}_{n^{1/\ell}} \otimes \bigotimes_{p=1}^{\ell-1} I + K^{+}_{n,\ell} \bigotimes_{p=1}^{i-1} I \otimes {G^{(\ell)}_{n^{1/\ell}}}^{T} \otimes \bigotimes_{p=1}^{\ell-1} I \right)$$

where $K^{\pm}_{m,\ell}$ have also a Kronecker tensor structure whenever the functions $\{\kappa_j\}_{j=1}^{\ell}$ are separable in the x_j variables.

Tensor Equations

A *different approach* that can be of interest is to use another structure.

Let us suppose that L_n is obtained as the discretization of a *multidimensional fractional operator*, i.e.,

$$L_n = \sum_{i=1}^{\ell} \left(K^{-}_{m,\ell} \bigotimes_{p=1}^{i-1} I \otimes G^{(\ell)}_{n^{1/\ell}} \otimes \bigotimes_{p=1}^{\ell-1} I + K^{+}_{n,\ell} \bigotimes_{p=1}^{i-1} I \otimes {G^{(\ell)}_{n^{1/\ell}}}^{T} \otimes \bigotimes_{p=1}^{\ell-1} I \right)$$

where $K^{\pm}_{m,\ell}$ have also a Kronecker tensor structure whenever the functions $\{\kappa_j\}_{j=1}^{\ell}$ are separable in the x_j variables.

The matrix: $\mathcal{M} = A_m \otimes I_n - hB_m \otimes L_n$ has now a lot of **redundant information**!

Tensor Equations: thou shalt compress!

As we have done for the *hierarchical formats*, we want

- A **compressed representation** of $\mathcal{M}$, possibly with a number of parameters that grows poly-logarithmically with the overall size…
- A **fast BLAS-like toolbox** to solve our problem in this format.

Tensor Equations: thou shalt compress!

As we have done for the *hierarchical formats*, we want

- A **compressed representation** of $\mathcal{M}$, possibly with a number of parameters that grows poly-logarithmically with the overall size...
- A **fast BLAS-like toolbox** to solve our problem in this format.

There exists *many formats* for which this is possible, *e.g.*, the CANDECOMP/PARAFAC (CP) decomposition, the Tucker format, the Tensor Train (TT), the TT-Tucker, *etc.*; see (Kolda and Bader 2009).

Tensor Equations: thou shalt compress!

As we have done for the *hierarchical formats*, we want

- A **compressed representation** of $\mathcal{M}$, possibly with a number of parameters that grows poly-logarithmically with the overall size...
- A **fast BLAS-like toolbox** to solve our problem in this format.

There exists *many formats* for which this is possible, *e.g.*, the CANDECOMP/PARAFAC (CP) decomposition, the Tucker format, the Tensor Train (TT), the TT-Tucker, *etc.*; see (Kolda and Bader 2009).

We focus on the **T**ensor-**T**rain format, since it has a simple enough toolbox to work with: TT-Toolbox.

Tensor-Train

But what is a *tensor*?

Tensor-Train

But what is a *tensor*?

A **tensor** is a **multidimensional array**, $a \in \mathbb{R}$ is a 0-tensor, $\mathbf{v} \in \mathbb{R}^{n_1}$ is a 1-tensor, $A \in \mathbb{R}^{n_1 \times n_2}$ is a 2-tensor, $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ is a 3-tensor, ...

Tensor-Train

But what is a *tensor*?

- A **tensor** is a **multidimensional array**, $a \in \mathbb{R}$ is a 0-tensor, $\mathbf{v} \in \mathbb{R}^{n_1}$ is a 1-tensor, $A \in \mathbb{R}^{n_1 \times n_2}$ is a 2-tensor, $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ is a 3-tensor, ...
- A **tensor** is a **multilinear maps** with respect to a fixed finite-dimensional $\mathbb{R}$ vector space V

$$\mathcal{A} : \underbrace{V^* \times \cdots \times V^*}_{p} \times \underbrace{V \times \cdots \times V}_{q} \to \mathbb{R},$$

Tensor-Train

But what is a *tensor*?

- A **tensor** is a **multidimensional array**, $a \in \mathbb{R}$ is a 0-tensor, $\mathbf{v} \in \mathbb{R}^{n_1}$ is a 1-tensor, $A \in \mathbb{R}^{n_1 \times n_2}$ is a 2-tensor, $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ is a 3-tensor, ...
- A **tensor** is a **multilinear maps** with respect to a fixed finite-dimensional $\mathbb{R}$ vector space V

$$\mathcal{A} : \underbrace{V^* \times \cdots \times V^*}_{p} \times \underbrace{V \times \cdots \times V}_{q} \to \mathbb{R},$$

- A **tensor** is an **element of the tensor product** of vector spaces

$$\mathcal{A} \in \underbrace{V \times \cdots \times V}_{p} \otimes \underbrace{V^* \otimes \cdots \otimes V^*}_{q}.$$

Tensor-Train

But what is a *tensor*?

- A **tensor** is a **multidimensional array**, $a \in \mathbb{R}$ is a 0-tensor, $\mathbf{v} \in \mathbb{R}^{n_1}$ is a 1-tensor, $A \in \mathbb{R}^{n_1 \times n_2}$ is a 2-tensor, $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ is a 3-tensor, ...
- A **tensor** is a **multilinear maps** with respect to a fixed finite-dimensional $\mathbb{R}$ vector space V

$$\mathcal{A} : \underbrace{V^* \times \cdots \times V^*}_{p} \times \underbrace{V \times \cdots \times V}_{q} \to \mathbb{R},$$

- A **tensor** is an **element of the tensor product** of vector spaces

$$\mathcal{A} \in \underbrace{V \times \cdots \times V}_{p} \otimes \underbrace{V^* \otimes \cdots \otimes V^*}_{q}.$$

The definition we select depends on the operations we want to perform.

Tensor-Train[1]

Let us start from trying to describe a *vector* associated with our discretization matrix $\mathcal{M}$.

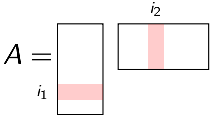
[1] For part of this material, a sincere thanks to Stefano Massei.

Tensor-Train[1]

Let us start from trying to describe a *vector* associated with our discretization matrix $\mathcal{M}$.

A rank-k matrix $A = U_1 U_2^T$ each entry is a dot product of vectors of length k

$$A(i_1, i_2) = U_1(i_1, :) \cdot U_2(:, i_2),$$



where the two indices select the left and right vectors.

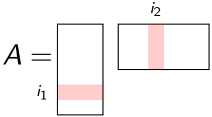
[1] For part of this material, a sincere thanks to Stefano Massei.

Tensor-Train[1]

Let us start from trying to describe a *vector* associated with our discretization matrix $\mathcal{M}$.

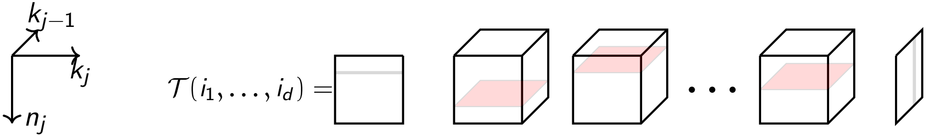
A rank-k matrix $A = U_1 U_2^T$ each entry is a dot product of vectors of length k

$$A(i_1, i_2) = U_1(i_1, :) \cdot U_2(:, i_2),$$



where the two indices select the left and right vectors. In a tensor of order d we insert $d-2$ matrices between the two vectors:

$$\mathcal{T}(i_1, \ldots, i_d) = U_1(i_1, :) \cdot U_2(:, :, i_2) \cdot \ \ldots \ \cdot U_{d-1}(:, :, i_{d-1}) \cdot U_d(:, i_d)$$

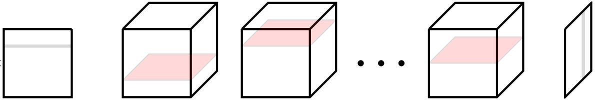


[1] For part of this material, a sincere thanks to Stefano Massei.

Tensor-Train

More formally, a tensor $\mathcal{T}$ is in TT decomposition if it can be written as

$\mathcal{T}(i_1, \ldots, i_d) =$



- Smallest possible tuple $(k_1, \ldots, k_{d-1})$ is called the **TT-rank** of $\mathcal{T}$.
- $U_j \in \mathbb{C}^{k_{j-1} \times n_j \times k_j}$ are called the **TT cores** of $\mathcal{T}$ (with $k_0 = k_d = 1$).
- If TT ranks are not large $\rightsquigarrow$ high compression ratio as d grows.
- TT decomposition multilinear with respect to the cores.

Tensor-Train

More formally, a tensor $\mathcal{T}$ is in TT decomposition if it can be written as

$$\mathcal{T}(i_1, \ldots, i_d) = \sum_{j_1=1}^{k_1} \cdots \sum_{j_{d-1}=1}^{k_{d-1}} U_1(i_1, j_1) U_2(j_1, i_2, j_2) \ldots U_d(j_{d-1}, i_d).$$

- Smallest possible tuple $(k_1, \ldots, k_{d-1})$ is called the **TT-rank** of $\mathcal{T}$.
- $U_j \in \mathbb{C}^{k_{j-1} \times n_j \times k_j}$ are called the **TT cores** of $\mathcal{T}$ (with $k_0 = k_d = 1$).
- If TT ranks are not large $\rightsquigarrow$ high compression ratio as d grows.
- TT decomposition multilinear with respect to the cores.

Tensor-Train

More formally, a tensor $\mathcal{T}$ is in TT decomposition if it can be written as

$\mathcal{T}(i_1, \ldots, i_d) =$ 

- Smallest possible tuple $(k_1, \ldots, k_{d-1})$ is called the **TT-rank** of $\mathcal{T}$.
- $U_j \in \mathbb{C}^{k_{j-1} \times n_j \times k_j}$ are called the **TT cores** of $\mathcal{T}$ (with $k_0 = k_d = 1$).
- If TT ranks are not large $\rightsquigarrow$ high compression ratio as d grows.
- TT decomposition multilinear with respect to the cores.

Tensor-Train

More formally, a tensor $\mathcal{T}$ is in TT decomposition if it can be written as

$$\mathcal{T}(i_1, \ldots, i_d) = \sum_{j_1=1}^{k_1} \cdots \sum_{j_{d-1}=1}^{k_{d-1}} U_1(i_1, j_1) U_2(j_1, i_2, j_2) \ldots U_d(j_{d-1}, i_d).$$

- Smallest possible tuple $(k_1, \ldots, k_{d-1})$ is called the **TT-rank** of $\mathcal{T}$.
- $U_j \in \mathbb{C}^{k_{j-1} \times n_j \times k_j}$ are called the **TT cores** of $\mathcal{T}$ (with $k_0 = k_d = 1$).
- If TT ranks are not large $\rightsquigarrow$ high compression ratio as d grows.
- TT decomposition multilinear with respect to the cores.

If for any $1 \leq \mu \leq d-1$ we group the first μ factors and last $d-\mu$ factors then

$$\mathcal{T}(i_1, \ldots, i_\mu, i_{\mu+1}, \ldots, i_d),$$

is the matrix-matrix product of two (large) matrices.

TT decomposition and matrix factorizations

The **μth unfolding** of $\mathcal{T} \in \mathcal{C}^{n_1 \times \cdots \times n_d}$ is obtained by arranging the entries in a matrix

$$T^{<\mu>} \in \mathbb{C}^{(n_1 \cdots n_\mu) \times (n_{\mu+1} \cdots n_d)}$$

where the corresponding index map is given by

$$\begin{aligned} \mathrm{ind} : \mathbb{N}^{n_1 \times \cdots \times n_d} &\to \mathbb{N}^{(n_1 \cdots n_\mu) \times (n_{\mu+1} \cdots n_d)} \\ \mathrm{ind}(i_1, \ldots, i_d) &= (i_{row}, i_{col}), \end{aligned}$$

where

$$i_{row} = 1 + \sum_{s=1}^{\mu} (i_s - 1) \prod_{t=1}^{s-1} n_t,$$

$$i_{col} = 1 + \sum_{s=\mu+1}^{d} (i_s - 1) \prod_{t=\mu+1}^{s-1} n_t$$

TT decomposition and matrix factorizations

We can compute the **compression** of the **tensor** by computing the SVD of the *unfoldings*.

Lemma (Oseledets 2011)

The **TT rank** of a tensor $\mathcal{T}$ is given by

$$\text{tt-rank}(\mathcal{T}) = (\text{rank}(T^{<1>}), \ldots, \text{rank}(T^{<d-1>})).$$

Input: Tensor $\mathcal{T}$, ranks $k_1, \ldots, k_d)$
Output: $U_1, \ldots, U_d$.
$k_0 = k_d = 1$;
for $\mu = 1, \ldots, d-1$ **do**
  Reshape $\mathcal{T}$ into $T^{<2>} \in \mathbb{C}^{k_{\mu-1} n_\mu \times (n_{\mu+1} \ldots n_d)}$;
  Compute rank-k_μ approximation $T^{<2>} \approx U \Sigma V^T$ (e.g. via SVD);
  Reshape U into $U_\mu \in \mathbb{C}^{k_{\mu-1} \times n_\mu \times k_\mu}$;
  Update $\mathcal{T}$ via $T^{<2>} \leftarrow U^T X^{<2>} = \Sigma V^T$;
end
Set $U_d = \mathcal{T}$;

Algorithm 1: $\text{TT-SVD}(\mathcal{T}, k_1, \ldots, k_d)$

The **proof** is obtained by simply following the steps of the algorithm.
We can use *tolerances* instead of fixed ranks.

TT decomposition and matrix factorizations

And we can estimate the resulting error using the best approximation properties of the SVD.

Theorem (Oseledets 2011)

Let $\mathcal{T}_{SVD}$ denote the tensor in TT decomposition obtained from TT-SVD. Then

$$\|\mathcal{T} - \mathcal{T}_{SVD}\| \leq \sqrt{\epsilon_1^2 + \cdots + \epsilon_d^2}$$

where

$$\epsilon_\mu^2 = \|T^{<\mu>} - U\Sigma V^T\|_F^2 = \sigma_{k_\mu+1}^2 + \sigma_{k_\mu+2}^2 + \ldots\ .$$

- We can modify the algorithm to accommodate different compression algorithms than the SVD,
- We can also compute the approximation via sketching algorithms, and avoiding using all the entries of $\mathcal{T}$.

TT-Matrices and matrix-vector products

If a **vector of length** $N = n_1 \times \ldots \times n_d$ is treated as a d–**dimensional tensor** with mode sizes n_k, and represented in TT–format, the **matrices acting on it** have the form

$$\mathcal{M}(i_1, \ldots, i_d, j_1, \ldots, j_d) = M_1(i_1, j_1) \ldots M(i_d, j_d), \qquad M_k(i_k, j_k) \in \mathbb{R}^{r_{k-1} \times r_k},$$

TT-Matrices and matrix-vector products

If a **vector of length** $N = n_1 \times \ldots \times n_d$ is treated as a d–**dimensional tensor** with mode sizes n_k, and represented in TT–format, the **matrices acting on it** have the form

$$\mathcal{M}(i_1, \ldots, i_d, j_1, \ldots, j_d) = M_1(i_1, j_1) \ldots M(i_d, j_d), \qquad M_k(i_k, j_k) \in \mathbb{R}^{r_{k-1} \times r_k},$$

- the first block indexes $i_1, \ldots, i_d$ enumerate the rows,

TT-Matrices and matrix-vector products

If a **vector of length** $N = n_1 \times \ldots \times n_d$ is treated as a d**–dimensional tensor** with mode sizes n_k, and represented in TT–format, the **matrices acting on it** have the form

$$\mathcal{M}(i_1,\ldots,i_d,j_1,\ldots,j_d) = M_1(i_1,j_1)\ldots M(i_d,j_d), \qquad M_k(i_k,j_k) \in \mathbb{R}^{r_{k-1}\times r_k},$$

- the first block indexes $i_1,\ldots,i_d$ enumerate the rows,
- the second block indexes $j_1,\ldots,j_d$ enumerate the columns.

Given $\mathcal{M}$ in TT–format, and a vector $\mathcal{X}$ in TT–format with cores X_k, and entries $X(j_1,\ldots,j_d)$ then the **matrix–vector multiplication** amounts to the following sum

$$\mathcal{Y}(i_1,\ldots,i_d) = \sum_{j_1,\ldots,j_d} \mathcal{M}(i_1,\ldots,i_d,j_1,\ldots,j_d)\mathcal{X}(j_1,\ldots,j_d) = Y_1(i_1)\ldots Y_d(i_d),$$

where $Y_k(i_k) = \sum_{j_k} M_k(i_k,j_k) \otimes X_k(j_k)$

TT-Matrices and matrix-vector products

If a **vector of length** $N = n_1 \times \ldots \times n_d$ is treated as a d**–dimensional tensor** with mode sizes n_k, and represented in TT–format, the **matrices acting on it** have the form

$$\mathcal{M}(i_1, \ldots, i_d, j_1, \ldots, j_d) = M_1(i_1, j_1) \ldots M(i_d, j_d), \qquad M_k(i_k, j_k) \in \mathbb{R}^{r_{k-1} \times r_k},$$

- the first block indexes $i_1, \ldots, i_d$ enumerate the rows,
- the second block indexes $j_1, \ldots, j_d$ enumerate the columns.

Given $\mathcal{M}$ in TT–format, and a vector $\mathcal{X}$ in TT–format with cores X_k, and entries $X(j_1, \ldots, j_d)$ then the **matrix–vector multiplication** amounts to the following sum

$$\mathcal{Y}(i_1, \ldots, i_d) = \sum_{j_1, \ldots, j_d} \mathcal{M}(i_1, \ldots, i_d, j_1, \ldots, j_d) \mathcal{X}(j_1, \ldots, j_d) = Y_1(i_1) \ldots Y_d(i_d),$$

where $Y_k(i_k) = \sum_{j_k} M_k(i_k, j_k) \otimes X_k(j_k)$ ⚠ The ranks of $\mathcal{Y}$ are the product of the ranks of the matrix and of the vector! So we need to **recompress** after every matrix-vector product.

TT-representation for our case

- We can use the same routine as before to *represent* the two BVM matrices,

```
%% Time-dependent operator
kval = 5;          % Grid power
m = 2^kval;        % Number of time
 ↪  steps
k = 2;
[Alpha,Beta] = mab(k,m);
A = Alpha(:,2:m+1);
B = Beta(:,2:m+1);
t0 = 0;
tf = 1;
h = (tf-t0)/m;
tA = tt_matrix(full(A),1e-14);
tA = tt_reshape(tA,2*ones(kval,2));
tB = tt_eye(2,kval);
```

TT-representation for our case

- We can use the same routine as before to *represent* the two BVM matrices,
- We build a tensor in which all the modes have size 2, this is usually called a Quantized-TT (QTT) formulation:

```
tA=tt_matrix(full(A),1e-14);
tA=tt_reshape(tA,2*ones(kval,2));
```

```
%% Time-dependent operator
kval = 5;          % Grid power
m = 2^kval;        % Number of time
↪  steps
k = 2;
[Alpha,Beta] = mab(k,m);
A = Alpha(:,2:m+1);
B = Beta(:,2:m+1);
t0 = 0;
tf = 1;
h = (tf-t0)/m;
tA = tt_matrix(full(A),1e-14);
tA = tt_reshape(tA,2*ones(kval,2));
tB = tt_eye(2,kval);
```

TT-representation for our case

- We can use the same routine as before to *represent* the two BVM matrices,
- We build a tensor in which all the modes have size 2, this is usually called a Quantized-TT (QTT) formulation:

```
tA=tt_matrix(full(A),1e-14);
tA=tt_reshape(tA,2*ones(kval,2));
```

- If we look at the values of k and maximal $\mathrm{tt\text{-}rank}$ we find:

k	2	3	4	5	6	7	8
$\max(\mathrm{tt\text{-}rank}(\mathcal{A}))$	3	5	6	7	7	7	9

```
%% Time-dependent operator
kval = 5;        % Grid power
m = 2^kval;      % Number of time
↪ steps
k = 2;
[Alpha,Beta] = mab(k,m);
A = Alpha(:,2:m+1);
B = Beta(:,2:m+1);
t0 = 0;
tf = 1;
h = (tf-t0)/m;
tA = tt_matrix(full(A),1e-14);
tA = tt_reshape(tA,2*ones(kval,2));
tB = tt_eye(2,kval);
```

TT-representation for our case

- We can act similarly also for the space operator.

```
%% Compression of the space part
tL = tt_matrix(L,1e-14);
tL = tt_reshape(tL,2*ones(kval+1,2));
tM = tt_eye(2,kval+1);
%% Final assembly
tMat = tkron(tA,tM)-h*tkron(tB,tL);
```

TT-representation for our case

- We can act similarly also for the space operator.
- We could be way more clever in the representation of these matrices, these are diagonal times Toeplitz, and we could do something specialized, *e.g.*, (Kazeev, Khoromskij, and Tyrtyshnikov 2013).

```
%% Compression of the space part
tL = tt_matrix(L,1e-14);
tL = tt_reshape(tL,2*ones(kval+1,2));
tM = tt_eye(2,kval+1);
%% Final assembly
tMat = tkron(tA,tM)-h*tkron(tB,tL);
```

TT-representation for our case

- We can act similarly also for the space operator.
- We could be way more clever in the representation of these matrices, these are diagonal times Toeplitz, and we could do something specialized, *e.g.*, (Kazeev, Khoromskij, and Tyrtyshnikov 2013).

```
%% Compression of the space part
tL = tt_matrix(L,1e-14);
tL = tt_reshape(tL,2*ones(kval+1,2));
tM = tt_eye(2,kval+1);
%% Final assembly
tMat = tkron(tA,tM)-h*tkron(tB,tL);
```

- Now that we have everything in this format, how can we solve our problem?

TT-representation for our case

- We can act similarly also for the space operator.
- We could be way more clever in the representation of these matrices, these are diagonal times Toeplitz, and we could do something specialized, *e.g.*, (Kazeev, Khoromskij, and Tyrtyshnikov 2013).

```
%% Compression of the space part
tL = tt_matrix(L,1e-14);
tL = tt_reshape(tL,2*ones(kval+1,2));
tM = tt_eye(2,kval+1);
%% Final assembly
tMat = tkron(tA,tM)-h*tkron(tB,tL);
```

- Now that we have everything in this format, how can we solve our problem?

TT-GMRES An option is to rephrase our favorite Krylov method using the TT arithmetic, (Dolgov 2013) and adapt what we know to build a preconditioner (Bertaccini and Durastante 2019).

TT-representation for our case

- We can act similarly also for the space operator.
- We could be way more clever in the representation of these matrices, these are diagonal times Toeplitz, and we could do something specialized, *e.g.*, (Kazeev, Khoromskij, and Tyrtyshnikov 2013).

```
%% Compression of the space part
tL = tt_matrix(L,1e-14);
tL = tt_reshape(tL,2*ones(kval+1,2));
tM = tt_eye(2,kval+1);
%% Final assembly
tMat = tkron(tA,tM)-h*tkron(tB,tL);
```

- Now that we have everything in this format, how can we solve our problem?

TT-GMRES An option is to rephrase our favorite Krylov method using the TT arithmetic, (Dolgov 2013) and adapt what we know to build a preconditioner (Bertaccini and Durastante 2019).

AMEn Use a *specialized solver* for linear systems in TT format (Dolgov and Savostyanov 2014).

✚ Concluding with an AMEn

Using AMEn (Dolgov and Savostyanov 2014) as

```
tx = amen_solve2(tMat,ttb,1e-6);
```

✚ Concluding with an AMEn

Using AMEn (Dolgov and Savostyanov 2014) as

```
tx = amen_solve2(tMat,ttb,1e-6);
```

k	m	n	IT	Residual	$\max(\text{tt-rank}(\mathcal{A}))$
2	64	128	9	2.231e-07	22
2	128	256	10	3.428e-07	26
2	256	512	14	5.925e-07	30
2	512	1024	22	3.957e-07	33
2	1024	2048	35	6.034e-07	37
2	2048	4096	47	6.968e-07	42

✚ Concluding with an AMEn

Using AMEn (Dolgov and Savostyanov 2014) as

```
tx = amen_solve2(tMat,ttb,1e-6);
```

k	m	n	IT	Residual	$\max(\text{tt-rank}(\mathcal{A}))$
3	64	128	8	2.252e-07	20
3	128	256	11	2.153e-07	24
3	256	512	15	2.138e-07	28
3	512	1024	18	2.950e-07	32
3	1024	2048	35	8.961e-07	36
3	2048	4096	50	3.821e-06	44

Concluding with an AMEn

Using AMEn (Dolgov and Savostyanov 2014) as

```
tx = amen_solve2(tMat,ttb,1e-6);
```

k	m	n	IT	Residual	$\max(\text{tt-rank}(\mathcal{A}))$
3	64	128	8	2.252e-07	20
3	128	256	11	2.153e-07	24
3	256	512	15	2.138e-07	28
3	512	1024	18	2.950e-07	32
3	1024	2048	35	8.961e-07	36
3	2048	4096	50	3.821e-06	44

- Behavior is *similar* to the matrix-equation solver,

✚ Concluding with an AMEn

Using AMEn (Dolgov and Savostyanov 2014) as

```
tx = amen_solve2(tMat,ttb,1e-6);
```

k	m	n	IT	Residual	$\max(\text{tt-rank}(\mathcal{A}))$
3	64	128	8	2.252e-07	20
3	128	256	11	2.153e-07	24
3	256	512	15	2.138e-07	28
3	512	1024	18	2.950e-07	32
3	1024	2048	35	8.961e-07	36
3	2048	4096	50	3.821e-06	44

- Behavior is *similar* to the matrix-equation solver,
- We could play around with different **settings** and **options** of the AMEn solver.

Concluding with an AMEn

Using AMEn (Dolgov and Savostyanov 2014) as

```
tx = amen_solve2(tMat,ttb,1e-6);
```

k	m	n	IT	Residual	$\max(\text{tt-rank}(\mathcal{A}))$
3	64	128	8	2.252e-07	20
3	128	256	11	2.153e-07	24
3	256	512	15	2.138e-07	28
3	512	1024	18	2.950e-07	32
3	1024	2048	35	8.961e-07	36
3	2048	4096	50	3.821e-06	44

- Behavior is *similar* to the matrix-equation solver,
- We could play around with different **settings** and **options** of the AMEn solver.
- Studying the right combination of parameters, representation, setups is still an open problem for the BVM all-at-once approaches.

Conclusion and summary

- We have seen how to work with linear multistep methods in boundary value form,
- We have discussed some structured preconditioning strategy for the resulting linear systems,
- We have introduced the machinery for working with tensor equations in the Tensor Train format.
- There are *many* open problems and possibilities to do better here.

Next up

- Fractional Laplacians,
- Rational approximations and matrix functions,
- A couple of applications to complex network theory.

Bibliography I

- Bertaccini, D. (2000). “A circulant preconditioner for the systems of LMF-based ODE codes”. In: *SIAM J. Sci. Comput.* 22.3, pp. 767–786. ISSN: 1064-8275. DOI: 10.1137/S1064827599353476. URL: https://doi.org/10.1137/S1064827599353476.
- — (2001). “Reliable preconditioned iterative linear solvers for some numerical integrators”. In: *Numer. Linear Algebra Appl.* 8.2, pp. 111–125. ISSN: 1070-5325. DOI: 10.1002/1099-1506(200103)8:2<111::AID-NLA234>3.0.CO;2-Q. URL: https://doi.org/10.1002/1099-1506(200103)8:2%3C111::AID-NLA234%3E3.0.CO;2-Q.
- Bertaccini, D. and F. Durastante (2019). “Block structured preconditioners in tensor form for the all-at-once solution of a finite volume fractional diffusion equation”. In: *Appl. Math. Lett.* 95, pp. 92–97. ISSN: 0893-9659. DOI: 10.1016/j.aml.2019.03.028. URL: https://doi.org/10.1016/j.aml.2019.03.028.
- Bertaccini, D. and F. Durastante (2018). “Limited memory block preconditioners for fast solution of fractional partial differential equations”. In: *J. Sci. Comput.* 77.2, pp. 950–970. ISSN: 0885-7474. DOI: 10.1007/s10915-018-0729-3. URL: https://doi.org/10.1007/s10915-018-0729-3.

Bibliography II

- Bertaccini, D. and M. K. Ng (2001). "The convergence rate of block preconditioned systems arising from LMF-based ODE codes". In: *BIT* 41.3, pp. 433–450. ISSN: 0006-3835. DOI: 10.1023/A:1021906926616. URL: https://doi.org/10.1023/A:1021906926616.
- Brugnano, L. and D. Trigiante (1998). *Solving differential problems by multistep initial and boundary value methods*. Vol. 6. Stability and Control: Theory, Methods and Applications. Gordon and Breach Science Publishers, Amsterdam, pp. xvi+418. ISBN: 90-5699-107-8.
- Dolgov, S. V. (2013). "TT-GMRES: solution to a linear system in the structured tensor format". In: *Russian J. Numer. Anal. Math. Modelling* 28.2, pp. 149–172. ISSN: 0927-6467. DOI: 10.1515/rnam-2013-0009. URL: https://doi.org/10.1515/rnam-2013-0009.
- Dolgov, S. V. and D. V. Savostyanov (2014). "Alternating minimal energy methods for linear systems in higher dimensions". In: *SIAM J. Sci. Comput.* 36.5, A2248–A2271. ISSN: 1064-8275. DOI: 10.1137/140953289. URL: https://doi.org/10.1137/140953289.
- Gu, X.-M. et al. (2015). "Strang-type preconditioners for solving fractional diffusion equations by boundary value methods". In: *J. Comput. Appl. Math.* 277, pp. 73–86. ISSN: 0377-0427. DOI: 10.1016/j.cam.2014.08.011. URL: https://doi.org/10.1016/j.cam.2014.08.011.

Bibliography III

Kazeev, V. A., B. N. Khoromskij, and E. E. Tyrtyshnikov (2013). "Multilevel Toeplitz matrices generated by tensor-structured vectors and convolution with logarithmic complexity". In: *SIAM J. Sci. Comput.* 35.3, A1511–A1536. ISSN: 1064-8275. DOI: 10.1137/110844830. URL: https://doi.org/10.1137/110844830.

Kolda, T. G. and B. W. Bader (2009). "Tensor decompositions and applications". In: *SIAM Rev.* 51.3, pp. 455–500. ISSN: 0036-1445. DOI: 10.1137/07070111X. URL: https://doi.org/10.1137/07070111X.

Oseledets, I. V. (2011). "Tensor-train decomposition". In: *SIAM J. Sci. Comput.* 33.5, pp. 2295–2317. ISSN: 1064-8275. DOI: 10.1137/090752286. URL: https://doi.org/10.1137/090752286.