

Fully Implicit Methods for Systems of PDEs

Åsmund Ødegård

Simula Research Laboratory

Outline

- Overview, Goal
- Operator splitting based methods
- Example with problem
- Overview Fully Implicit method
- Nonlinear problem
- Examples

Overview and Goal

- In an earlier talk, Tom discussed implementations for Systems of PDEs solved by splitting-techniques.
- We focus on Implicit methods for Systems of PDEs.
- Jacobi or Gauss–Seidel based solution strategies may be either slow or unstable, or both.
- Main goal: Establish framework for implementation of fully implicit solvers.
- Retain the flexibility and extensibility of the operator splitting approach.
- Talk based on chapter submitted to the next Diffpack book.

Re-cap: Operator splitting

Consider systems of differential equations on the form

$$L_1(u, v) = f, \quad (1)$$

$$L_2(u, v) = g, \quad (2)$$

Given u^n, v^n , we can formulate two different iterations:

- Jacobi:
 1. Compute u^{n+1} such that $L_1(u^{n+1}, v^n) = f$.
 2. Then compute v^{n+1} such that $L_2(u^n, v^{n+1}) = g$.
- Gauss–Seidel:
 1. Compute u^{n+1} such that $L_1(u^{n+1}, v^n) = f$.
 2. Compute v^{n+1} such that $L_2(u^{n+1}, v^{n+1}) = g$.
- System reduced to solving two scalar equations

Splitting is not safe

Consider the system:

$$\begin{aligned}y' &= z^2 & y(0) &= -\varepsilon, \\z' &= -y^2 & z(0) &= \varepsilon.\end{aligned}\tag{3}$$

The system may easily be reformulated as:

$$\begin{aligned}y' &= y^2, & y(0) &= -\varepsilon, \\z' &= -z^2, & z(0) &= \varepsilon.\end{aligned}\tag{4}$$

Analytical solution is:

$$y(t) = \frac{-\varepsilon}{1 + t\varepsilon}, \quad z(t) = \frac{\varepsilon}{1 + t\varepsilon}.\tag{5}$$

Gauss–Seidel on ODE problem

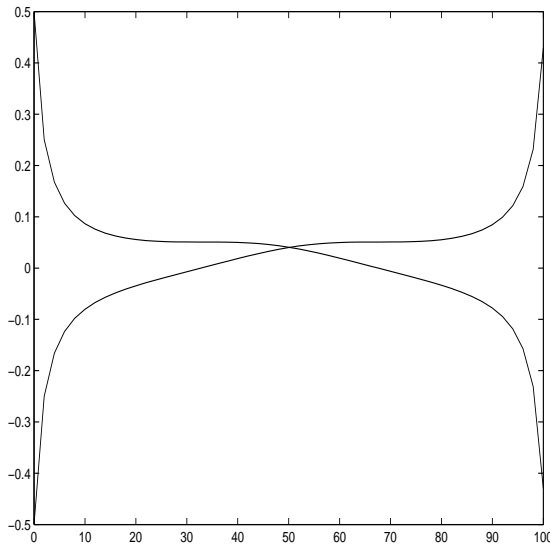
The Gauss–Seidel method can be formulated in two ways for the *original* system (3), depending on in which order we solve the equations; either

$$\begin{aligned} y_{n+1} &= y_n + \Delta t z_n^2 & y_0 &= -\varepsilon, \\ z_{n+1} &= z_n - \Delta t y_{n+1}^2 & z_0 &= \varepsilon, \end{aligned} \tag{6}$$

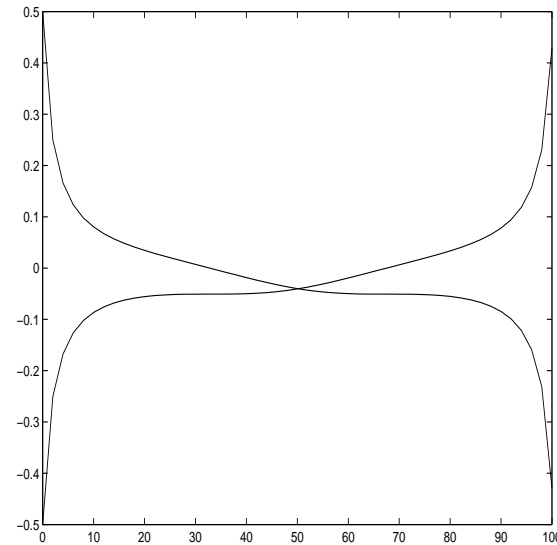
or

$$\begin{aligned} z_{n+1} &= z_n - \Delta t y_n^2 & y_0 &= -\varepsilon, \\ y_{n+1} &= y_n + \Delta t z_{n+1}^2 & z_0 &= \varepsilon. \end{aligned} \tag{7}$$

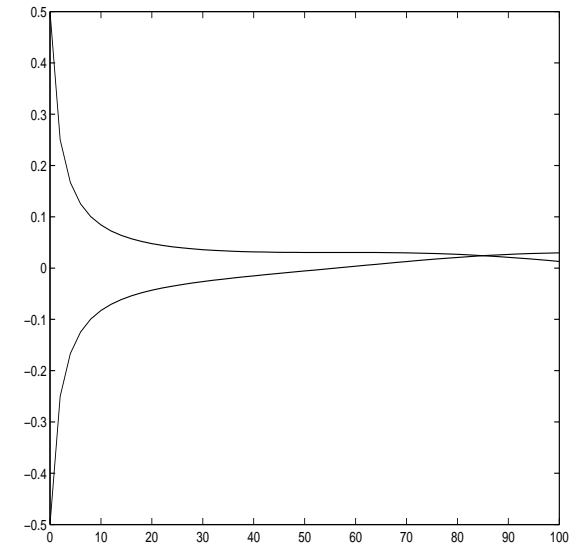
These methods give the solution curves plotted in the figures on the next slide



(a) Solution of system (6), with $\Delta t = 0.0014$ and $\varepsilon = 0.5$.



(b) Solution of system (7), with $\Delta t = 0.0014$ and $\varepsilon = 0.5$.



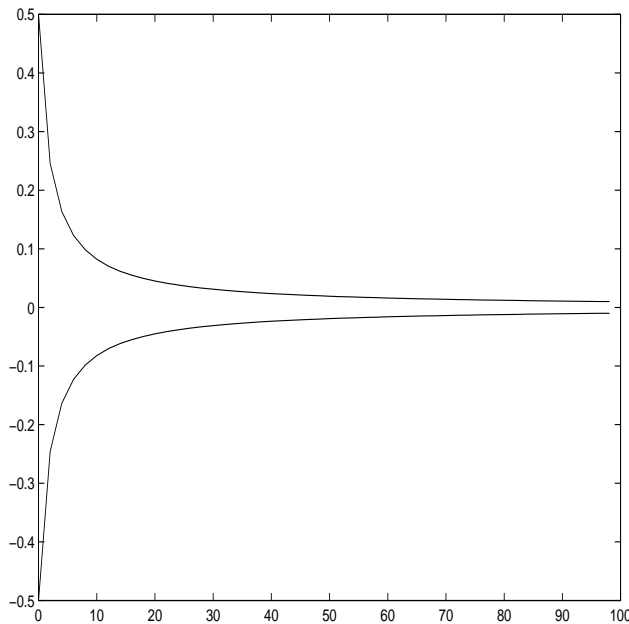
(c) Solution of system (6), with $\Delta t = 0.0003$ and $\varepsilon = 0.5$.

The solution of (3) approximated with Gauss–Seidel’s method.

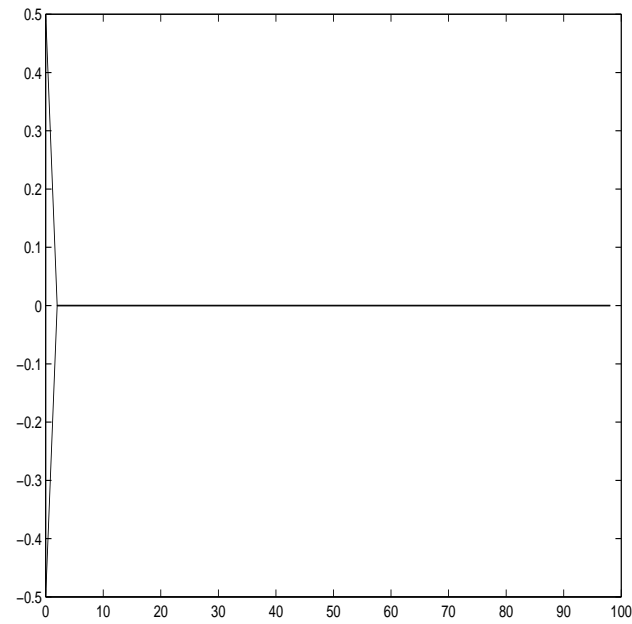
Jacobi on the ODE problem

$$\begin{aligned}y_{n+1} &= y_n + \Delta t z_n^2 & y_0 &= -\varepsilon, \\z_{n+1} &= z_n - \Delta t y_n^2 & z_0 &= \varepsilon.\end{aligned}$$

(8)



(d) Stability-
condition fulfilled,
 $\Delta t = 1$ and $\varepsilon = 0.5$.



(e) Stability-
condition not
satisfied, $\Delta t = 2.002$,
and $\varepsilon = 0.5$.

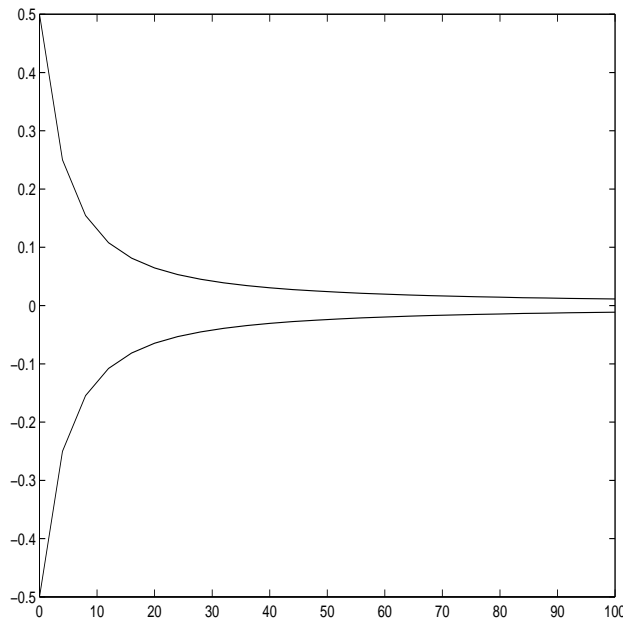
Fully Implicit solver

$$y_{n+1} = y_n + \Delta t z_{n+1}^2$$
$$z_{n+1} = z_n - \Delta t y_{n+1}^2$$

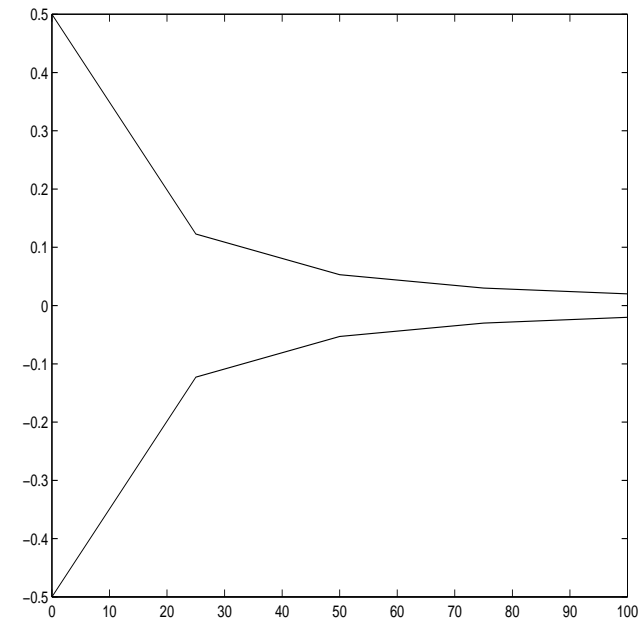
$$y_0 = -\varepsilon,$$

$$z_0 = \varepsilon.$$

(9)



(f) $\Delta t = 4, \varepsilon = 0.5$



(g) $\Delta t = 25, \varepsilon = 0.5$

Lessons learned so far...

- Simple splitting–techniques may lead to severe difficulties.
- The order of the equations in Gauss–Seidel may affect the solution.
- Simple splitting–techniques may converge to the wrong solution.
- Fully implicit method are more safe.

Fully Implicit Implementation

Consider the system

$$\begin{aligned} -\nabla^2 u + u + v &= f, \\ -\nabla^2 v + u + v &= g, \end{aligned} \tag{10}$$

A: $A_{i,j} = (\nabla N_i, \nabla N_j)$ and **M**: $M_{i,j} = (N_i, N_j)$.

Finite element form of system (10) written with block matrices:

$$\begin{bmatrix} \mathbf{A} + \mathbf{M} & \mathbf{M} \\ \mathbf{M} & \mathbf{A} + \mathbf{M} \end{bmatrix} \begin{bmatrix} \hat{\mathbf{u}} \\ \hat{\mathbf{v}} \end{bmatrix} = \begin{bmatrix} \mathbf{F} \\ \mathbf{G} \end{bmatrix}, \tag{11}$$

where $F_j = (f, N_j)$ and $G_j = (g, N_j)$.

Fully Implicit cont...

The Gauss–Seidel iteration written with block matrices:

$$\begin{aligned} [\mathbf{A} + \mathbf{M}] \hat{\mathbf{u}}^{(k+1)} &= -\mathbf{M}\hat{\mathbf{v}}^{(k)} + \mathbf{F}, \\ [\mathbf{A} + \mathbf{M}] \hat{\mathbf{v}}^{(k+1)} &= -\mathbf{M}\hat{\mathbf{u}}^{(k+1)} + \mathbf{G}. \end{aligned} \tag{12}$$

- The Gauss–Seidel method and the implicit method only differ in how terms are arranged.
- If we have stand–alone solvers for the sub problems, these can be reused as before.
- We need to rearrange the terms, by changing the assembling of the linear system.

Code Framework

- Manager-class derived from `SystemFEM`.
- `SystemFEM` have pointer to each sub-solver.
- Storage for full system.
- Implements a new `MakeSystem` which assemble the complete system.
- Based on framework for Mixed Methods, `MxFEM`.
- Sub-solvers must derive the `SimulatorSystem` base-class.
- Sub-solvers must implement the `integrandsMX` method.
- This is not in Diffpack yet.

Nonlinear Problems

- We use the Newton–Raphson method.
- The Newton–iteration for $F(u) = 0$:

$$\frac{\partial F}{\partial u}(u_{i-1})r_{i;u} = -F(u_{i-1}), \quad (13)$$

- Nonlinear system. Given to non–linear problems:

$$F_1(u) = 0 \quad \text{and} \quad F_2(v) = 0 \quad (14)$$

- Assume that Diffpack solvers exist.
- Consider coupled problem:

$$\begin{aligned} F_1(u, v) &= 0, \\ F_2(u, v) &= 0. \end{aligned} \quad (15)$$

Gauss–Seidel for Nonlinear Problem

- Gauss–Seidel iteration:

$$\begin{aligned}F_1(u_k, v_{k-1}) &= 0, \\F_2(u_k, v_k) &= 0.\end{aligned}\tag{16}$$

- The corresponding Newton iterations on the discrete problem:

$$\begin{aligned}\frac{\partial F_1}{\partial u}(u_k^{(i-1)}, v_{k-1})r_{i;u} &= -F_1(u_k^{(i-1)}, v_{k-1}), \\ \frac{\partial F_2}{\partial v}(u_k, v_k^{(i-1)})r_{i;v} &= -F_2(u_k, v_k^{(i-1)}).\end{aligned}\tag{17}$$

- Contain only terms present in the sub-solvers.

Fully Implicit solver

- Apply Newton method to (15) directly:

$$\begin{bmatrix} \frac{\partial F_1^{(i-1)}}{\partial u} & \frac{\partial F_1^{(i-1)}}{\partial v} \\ \frac{\partial F_2^{(i-1)}}{\partial u} & \frac{\partial F_2^{(i-1)}}{\partial v} \end{bmatrix} \begin{bmatrix} r_{i;u} \\ r_{i;v} \end{bmatrix} = \begin{bmatrix} -F_1^{(i-1)} \\ -F_2^{(i-1)} \end{bmatrix} \quad (18)$$

- The Jacobi matrices off-diagonal are new.
- The rest of the terms are present in the Gauss–Seidel solver.
- This is different from the linear case.
- Still, the resulting linear system is similar to what we obtain in the linear case.
- We can implement a fully implicit solver as we have done in the linear case.

Examples

Pipeflow problem

$$\begin{aligned}\nabla \cdot [\mu \nabla w] &= -\beta, \\ \nabla^2 T &= -\kappa^{-1} \mu \dot{\gamma}^2, \\ w &= 0 \text{ on } \partial\Omega, \\ T &= T_0 \text{ on } \partial\Omega_1, \quad T = T_1 \text{ on } \partial\Omega_2.\end{aligned}\tag{19}$$

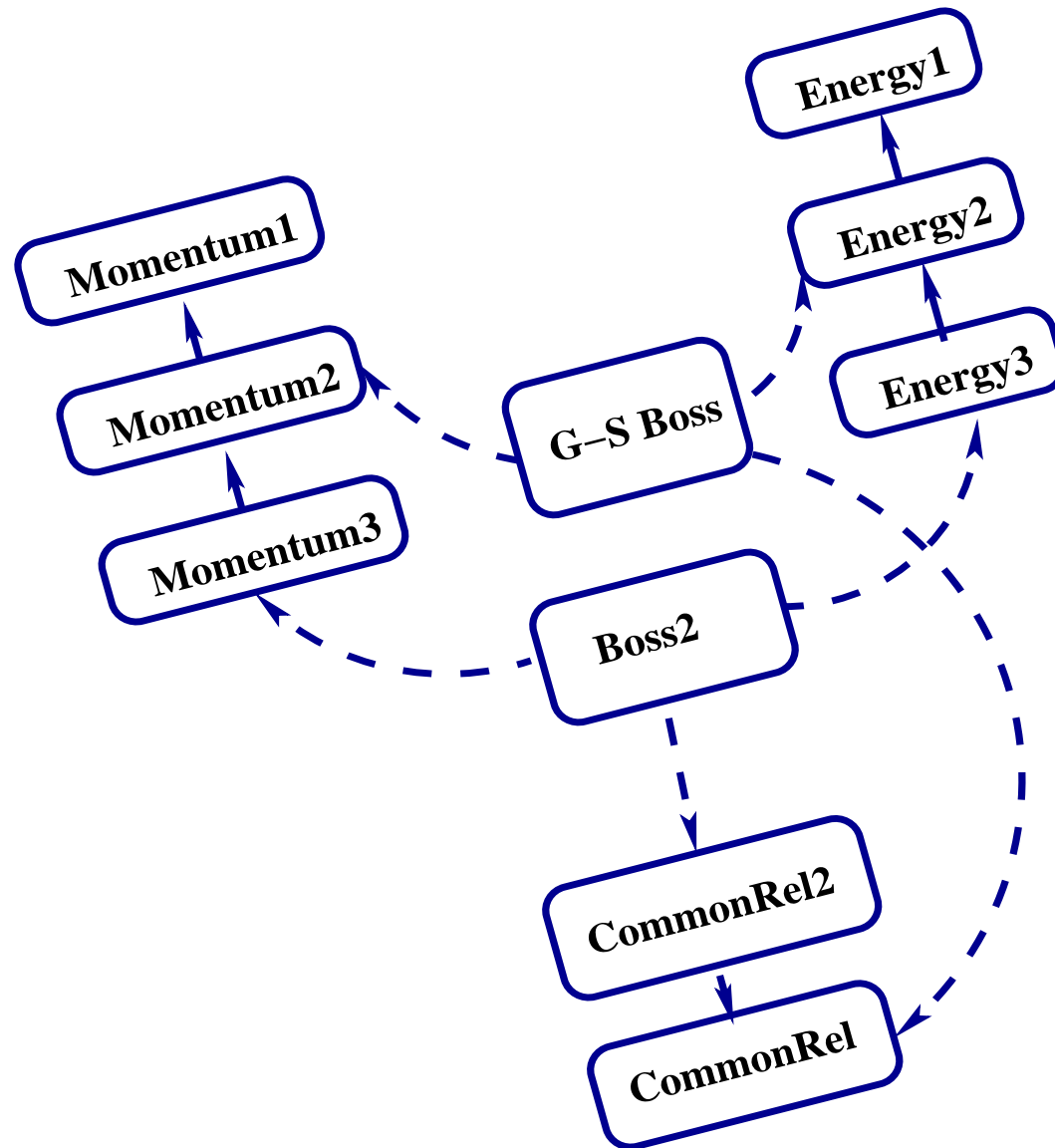
Here,

$$\mu = \mu_T(T) \mu_w(\dot{\gamma}),$$

and

$$\dot{\gamma} = \sqrt{(w_{,x})^2 + (w_{,y})^2},$$

Pipeflow classes



Pipeflow cont.

- Momentum3 and Energy3 implement `integrandsMx` and `fillEssBC` as well as some book-keeping functions.
- `CommonRe12` redefine functions which use the `FiniteElement`, since we are using the Mixed framework.

pow.law exponent:	$n = 0.8$		$n = 1.0$	
	iter.	time	iter.	time
Fully Implicit	4	4342.7s	3	3487.6s
Gauss–Seidel	3	7747.8s	3	5995.8s

Performance comparison. *iter* is Newton iterations in the fully implicit case and outer iterations in the Gauss–Seidel case.