Efficient Implementation of Quasi-Newton Methods for Partitioned Fluid-Structure Simulations

Klaudius Scheufele

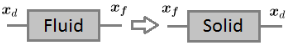
04.05.2015

Contents

Partitioned Fluid-Structure Simulations

- two physical domains, i. e., **fluid** and **solid**, different models
- coupling conditions at fluid-structure interface
- **partitioned approach:** black-box field solvers

$$\boldsymbol{F} : \boldsymbol{x}_d \mapsto \boldsymbol{x}_f$$
$$\boldsymbol{S} : \boldsymbol{x}_f \mapsto \boldsymbol{x}_d .$$



- instabilities → *coupling algorithms*

$$\textbf{coupling scheme} = \text{coupling system (fixed-point formulation)} + \text{post processing method}$$

- implicit coupling scheme → iterative solution

Coupling Schemes: Post Processing

accelerate and stabilize pure fixed-point iteration
$\rightarrow$ constant underrelaxation, Aitken, **quasi-Newton methods**.

$$x^{k+1} = \tilde{x}^k - \widehat{J}_{\tilde{R}}^{-1}(x^k)\left(H(x^k) - x^k\right)$$

- low-rank update in each iteration based on input/output differences

$$W_k = \left[\Delta\tilde{x}_0^k, \Delta\tilde{x}_1^k, \cdots, \Delta\tilde{x}_{k-1}^k\right], \qquad \text{with } \Delta H_i^k = \Delta\tilde{x}_i^k = \tilde{x}^k - \tilde{x}^i$$
$$V_k = \left[\Delta R_0^k, \Delta R_1^k, \cdots, \Delta R_{k-1}^k\right], \qquad \text{with } \Delta R_i^k = R(x^k) - R(x^i).$$

- *secant equation* for the inverse system Jacobian

$$\widehat{J}_{\tilde{R}}^{-1}(x^k)V_k = W_k$$

- add a minimization condition (to obtain uniqueness)

$$\left\|\widehat{J_{\tilde{R}}}^{-1}(\tilde{x}^k)\right\|_F \rightarrow \min \quad or \quad \left\|\widehat{J_{\tilde{R}}}^{-1}(\tilde{x}^k) - \widehat{J_{\tilde{R}}}_{prev}^{-1}\right\|_F \rightarrow \min$$

Type I: $\widehat{J_{\tilde{R}}}^{-1}(\tilde{x}^k) = W_k\left(V_k^T V_k\right)^{-1} V_k^T$ IQN-ILS

Type II: $\widehat{J_{\tilde{R}}}^{-1}(\tilde{x}^k) = \widehat{J_{\tilde{R}}}_{prev}^{-1} + \left(W_k - \widehat{J_{\tilde{R}}}_{prev}^{-1} V_k\right)\left(V_k^T V_k\right)^{-1} V_k^T$ IQN-IMVJ

Jacobian Update - Solving the Normal Equations

$$x^{k+1} = \tilde{x}^k - \mathcal{J} R(x^k)$$

Two types of update formulas:

$$\mathcal{J} = W \left(V^T V\right)^{-1} V^T \qquad \text{(matrix-free)}$$

$$\mathcal{J} = \mathcal{J}_{prev} + (W - \mathcal{J}_{prev} V) \left(V^T V\right)^{-1} V^T \qquad \text{(store Jacobian)}$$

with $\mathcal{J} \in \mathbb{R}^{n \times n}$, $V, W \in \mathbb{R}^{n \times m}$, and $m \ll n$

solving the normal equations:

$$\left(V^T V\right)^{-1} \rightsquigarrow \left(V^T V\right) z = V^T y$$

for arbitrary y and corresponding z.

Type I: $\alpha = ZR(x^k)$, i. e., $y = R(x^k)$ and $\alpha = z$

Type II: $Z = \left(V^T V\right)^{-1} V^T \in \mathbb{R}^{m \times n}$, i. e., $y_i \in \{e_1, \cdots, e_n\}$ and $Z = (z_i)$

usually V close to rank-deficient $\rightarrow$ product $V^T V$ doubles (bad) condition number

Jacobian Update - QR-Decoposition

alternative problem formulation

$$\mathbf{min}_{z\in\mathbb{R}^n} \left\| Vz - y \right\|_2$$

better condition number, numerically more stable

- suffices to solve $Vz = y \rightarrow$ QR-dec of V, i. e., $V = QU$ with $Q \in \mathbb{R}^{n\times n}$, $U \in \mathbb{R}^{n\times m}$
- compute z from lin. system $\tilde{U}z = \tilde{Q}^T y$
  Type I: $y = R(x^k)$ and $\alpha = z$
  Type II: $z = \left(V^T V\right)^{-1} V^T \in \mathbb{R}^{m\times n}$, i. e., $y_i \in \{e_1, \cdots, e_n\}$ and $Z = (z_i)$

solve $\boldsymbol{Vz} = \boldsymbol{y}$ via QR-dec.

compute $\boldsymbol{QU} = \boldsymbol{V}$ $\in \frac{4}{3}\mathcal{O}(m^2 n)$
for $i = 1$ *to* n **do**
  backw. subst. $\tilde{\boldsymbol{U}}\boldsymbol{z} = \boldsymbol{Q}^T(\cdot, i)$ $\in \frac{1}{2}\mathcal{O}(m^2)$
  $\boldsymbol{Z}(\cdot, i) = \boldsymbol{z}$
end

- requires $2m^2(n - \frac{m}{3}) \in \mathcal{O}(m^2 n)$ flops
- recomputed in every iteration
- but $V_k = [V_{k-1}, v]$, $v = \Delta R^k_{k-1}$
- **update QR-dec in $\mathcal{O}(mn)$ rather than re-compute**

Update the QR-Decomposition

$V_k = \left[\tilde{V}_{k-1}, v\right] \rightarrow$ required functionality: *insertColumn(vec v, index k)*
deleteColumn(index k)

delete Column:

$V = (V_1, v, V_2) = Q(R_1, r, R_2)$ and $v = Qr$, $\bar{V} = (V_1, V_2) = Q(R_1, R_2) = Q\hat{R}$

- choose Givens matrices to annihilate subdiagonal elements of R
  $H\hat{R} = H_{n-1,n} \cdots H_{k+1,k+2} H_{k,k+1} \hat{R} = \begin{pmatrix} \bar{R} \\ 0 \end{pmatrix}$
- then $QH^T = QH_{k,k+1} \cdots H_{n-1,n}\hat{R} = (\bar{Q}, \bar{q})$ orthonormal
- $\rightarrow \bar{V} = \bar{Q}\bar{R}$

$$\hat{R} = \begin{bmatrix} * & * & * & * \\ 0 & * & * & * \\ 0 & 0 & * & * \\ 0 & 0 & \ominus & * \\ 0 & 0 & 0 & \ominus \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Update the QR-Decomposition

insert Column:

$V = (V_1, V_2) = Q(R_1, R_2) = QR \in \mathbb{R}^{n\times(m-1)}$

insert $v \in \mathbb{R}^n$ between V_1 and V_2, i. e., $\bar(V) = (V_1, v, V_2) = (Q, v)\begin{pmatrix} R_1 & 0R_2 \\ 0^T10^T & \end{pmatrix}$

- step 1: apply Gram-Schmidt process to obtain
  $(Q, v) = (Q, q)\begin{pmatrix} I & r \\ 0 & \rho \end{pmatrix}, \quad Q^Tq = 0, \quad \|q\| = 1.$
- then $\bar{V} = (Q, q)\begin{pmatrix} R_1 & r & R_2 \\ 0^T & \rho & 0^T \end{pmatrix} = \tilde{Q}\tilde{R}$
- choose Givens matrices so that $\bar{R}$ is upper triangular
  $G\tilde{R} = G_{k,k+1}\cdots G_{n-1,n}\tilde{R} = \bar{R}$
- then $QH^T = QH_{k,k+1}\cdots H_{n-1,n}\hat{R} = (\bar{Q}, \bar{q})$ orthonormal
- then $\tilde{Q}G^T = \tilde{Q}G_{n-1,n}\cdots G_{k,k+1} = \bar{Q}$ and $\rightarrow \ \bar{A} = \bar{Q}\bar{R}$

$$\tilde{R} = \begin{bmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & \ominus & \oplus & * & * \\ 0 & \ominus & 0 & \oplus & * \\ 0 & \ominus & 0 & 0 & \oplus \\ 0 & \ominus & 0 & 0 & 0 \\ 0 & \ominus & 0 & 0 & 0 \\ 0 & \ominus & 0 & 0 & 0 \end{bmatrix}$$

Parallelization of Updated QR-dec

interface unknowns are decomposed and distributed on N processors, i, e.,

$$x = (\text{proc}\,\sharp 1, \text{proc}\,\sharp 2, \cdots, \text{proc}\,\sharp N)^T \in \mathbb{R}^n$$

Thus, matrices V, W, Q are decomposed and distributed block-row wise:

$$V, W = \begin{pmatrix} \text{proc}\,A \\ \text{proc}\,B \\ \vdots \\ \text{proc}\,X \end{pmatrix} \in \mathbb{R}^{n \times m}, \quad Q = \begin{pmatrix} \text{proc}\,A \\ \text{proc}\,B \\ \vdots \\ \text{proc}\,X \end{pmatrix} \in \mathbb{R}^{n \times n}$$

- each proc holds copy of $R \in \mathbb{R}^{m \times m}$, and local row-block of Q.
- all procs compute the same Givens rotations and update R and their part of Q
  $\rightarrow$ embarrassingly parallel

Parallelization IQN-ILS

parallelize QN-update

$$\mathcal{J} = W_k \left(V_k^T V_k \right)^{-1} V_k^T$$
$$x^{k+1} = \tilde{x}^k + W_k \underbrace{\left(V_k^T V_k \right)^{-1} V_k^T \left(-R(x^k) \right)}_{\alpha}$$

compute $R\alpha = Q^T \left(-R(x^k) \right)$ via updated QR-dec in parallel:

- every proc computes his part of rhs
- all-reduce (sum up) to master, master computes α, broadcast

Parallelization IQN-IMVJ

parallelize QN-update

$$\mathcal{J} = \mathcal{J}_{prev} + (W_k - \mathcal{J}_{prev} V_k)\left(V_k^T V_k\right)^{-1} V_k^T$$
$$x^{k+1} = \tilde{x}^k + \left(\mathcal{J}_{prev} + (W_k - \mathcal{J}_{prev} V_k)\left(V_k^T V_k\right)^{-1} V_k^T\right)\left(-R(x^k)\right)$$

compute $Z = (z_1, \cdots, z_n) = \left(V_k^T V_k\right)^{-1} V_k^T$ from $Rz_i = Q^T(i)$ via updated QR-dec in parallel

Z is decomposed block-column wise, i. e., $z = \begin{pmatrix} [\text{proc A}] & [\text{proc B}] & \cdots & [\text{proc X}] \end{pmatrix}$

Problem:

- $\mathcal{J}$ arbitrary structure, dense $\rightarrow$ decomposition/distribution unclear
- expensive multiplications $\mathcal{J}_{prev} V_k$ and $\tilde{W}Z$, $\quad \tilde{W} = (W - \mathcal{J}_{prev} V)$

Parallelization IQN-IMVJ

first idea: Store old representations $\widetilde{W}_{k-1} := (W - \mathcal{J}_{k-1} V_k)$, V_{k-1} and Z_{k-1}.

$$\text{update} \quad \widetilde{W}_k := W_k - \underbrace{\widetilde{W}_{k-1} Z_{k-1}}_{\mathcal{J}_{k-1} - \mathcal{J}_{k-2}} V_k$$

However, the addition of $\mathcal{J}_{k-1}$ induces some problems:

$$\mathcal{J}_0 = 0, \widetilde{W}_0 = 0, Z_0 = 0, V_0 = 0;$$

$$\begin{array}{ll}
\widetilde{W}_1 := W_1 - \widetilde{W}_0 Z_0 V_1 = W_1 & \mathcal{J}_1 = 0 + \widetilde{W}_1 Z_1 V_1^T \\
\widetilde{W}_2 := W_2 - \widetilde{W}_1 Z_1 V_2 & \mathcal{J}_2 = \mathcal{J}_1 + \widetilde{W}_2 Z_2 = \widetilde{W}_1 Z_1 + \widetilde{W}_2 Z_2 \\
\widetilde{W}_3 := W_3 - (\widetilde{W}_1 Z_1 + \widetilde{W}_2 Z_2) V_3 & \mathcal{J}_3 = \mathcal{J}_2 + \widetilde{W}_3 Z_3 \\
\vdots & \vdots \\
\widetilde{W}_k := W_k - \left(\sum_{i=1}^{k-1} \widetilde{W}_i Z_i \right) V_k & \mathcal{J}_k = \sum_{i=1}^{k} \widetilde{W}_i Z_i
\end{array}$$

Hence, we need to store $\widetilde{W}_1, \cdots, \widetilde{W}_{k-1}, Z_1, \cdots, Z_{k-1}$ and $V_k \ni V_1, \cdots, V_{k-1}$ over all time steps

Parallelization IQN-IMVJ

$$\mathcal{J} = \mathcal{J}_{prev} + (W_k - \mathcal{J}_{prev} V_k) \left(V_k^T V_k \right)^{-1} V_k^T$$

attempt for decomposition:
decompose and distribute $\mathcal{J}$ block-column wise

$$\underbrace{\left(\begin{array}{c|c|c} A & \cdots & D \end{array} \right)}_{\mathcal{J}prev} + \left(\underbrace{\begin{pmatrix} A \\ \vdots \\ D \end{pmatrix}}_{W} - \underbrace{\left(\begin{array}{c|c|c} A & \cdots & D \end{array} \right)}_{\mathcal{J}prev} \underbrace{\begin{pmatrix} A \\ \vdots \\ D \end{pmatrix}}_{V} \right) \underbrace{\left(\begin{array}{c|c|c} A & \cdots & D \end{array} \right)}_{Z}$$

multiply $\mathcal{J}_{prev} V$: $\left(\begin{array}{c|c|c|c} A_1 & B_1 & C_1 & D_1 \\ A_2 & B_2 & C_2 & D_2 \\ A_3 & B_3 & C_3 & D_3 \\ A_4 & B_4 & C_4 & D_4 \end{array} \right) \begin{pmatrix} A \\ B \\ C \\ D \end{pmatrix} = \begin{pmatrix} A_1 A + B_1 B + C_1 C + D_1 D \\ A_2 A + B_2 B + C_2 C + D_2 D \\ A_3 A + B_3 B + C_3 C + D_3 D \\ A_4 A + B_4 B + C_4 C + D_4 D \end{pmatrix} \begin{array}{l} \to A \\ \to B \\ \to C \\ \to D \end{array}$

multiply $\tilde{W} Z$: $\begin{pmatrix} A \\ B \\ C \\ D \end{pmatrix} \left(\begin{array}{c|c|c|c} A & B & C & D \end{array} \right) = \left(\begin{array}{c|c|c|c} AA^{(1)} & AB^{(4)} & AC^{(3)} & AD^{(2)} \\ BA^{(2)} & BB^{(1)} & BC^{(4)} & BD^{(3)} \\ CA^{(3)} & CB^{(2)} & CC^{(1)} & CD^{(4)} \\ DA^{(4)} & DB^{(3)} & DC^{(2)} & DD^{(1)} \\ \downarrow & \downarrow & \downarrow & \downarrow \\ A & B & C & D \end{array} \right)$

cyclic send-receive operation: after step (1): $A_z \to B_w$, $B_z \to C_w$, $C_z \to D_w$, $D_z \to A_w$ usw. ...