Implementation Advanced Multi-Vector Variants (efficient update/ subspace tracking)

Klaudius Scheufele

March 31, 2016

1 Implementation IMVJ Restart Approach

Solve non-linear fixed-point equation

$$H(x) = x \quad \Leftrightarrow \quad \tilde{R}(\tilde{x}) = \tilde{x} - H^{-1}(\tilde{x}) = H(x) - x \stackrel{!}{=} 0 \tag{1}$$

by means of (quasi-)Newton iterations:

$$\begin{aligned} x^{k+1} &= H(x^k) - \left(J_{prev}^{-1} + (W - J_{prev}^{-1}V)(V^TV)^{-1}V^T\right) R(x^k) \\ &= \tilde{x}^k - \left(J_{prev}^{-1} + \widetilde{W}_k V_k^\dagger\right) r^k \\ &= \tilde{x}^k - \left(\sum_{q=0}^{n} \widetilde{W}_k V_k^\dagger\right) r^k \end{aligned}$$

1.1 General Algorithm

Store matrices

$$\begin{aligned} \widetilde{W}_k &= (W_k - J_{prev}^{-1}V_k) \\ V_k^\dagger &= (V^TV)^{-1}V^T, \qquad k = 1, \ldots, n \end{aligned}$$

and compute (quasi-)Newton update without building Jacobian matric explicitly using only matrix-vector products

$$x^{k+1} = \tilde{x}^k - \sum_{q=0}^{n} \widetilde{W}_k \left(V_k^\dagger \cdot r^k\right) \tag{2}$$

1.) compute $\widetilde{W}_k^n := (W_k^n - \sum_{q=0}^{n} \widetilde{W}_{k_q} \left(V_{k_q}^\dagger \cdot r^k\right)$

2.) compute $\Delta x = \tilde{x}^k - \sum_{q=0}^{n-1} \widetilde{W}_{k_q} \left(V_{k_q}^\dagger \cdot r^{k_n}\right) - \widetilde{W}_{k_n} \left(V_{k_n}^\dagger \cdot r^{k_n}\right)$

3.) if time step converged, store matrices $\widetilde{W}_{k_n}, V_{k_n}^\dagger$. Partition simulation in chunks of M time steps and store at most matrices from the last M time steps.

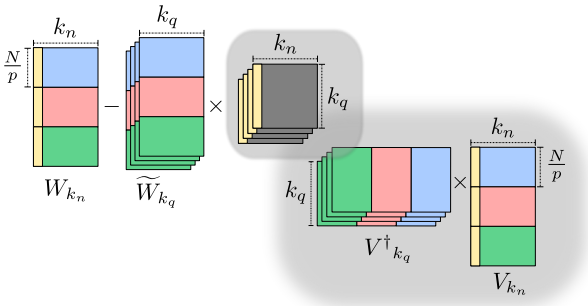
4.) after M time steps, restart.

1.2 Update $\widetilde{W}$

In each iteration, $\widetilde{W}_{k_n}$ changes only slightly, depending on the new columns in W_k and V_k. Hence, update $\widetilde{W}_k$ from $\widetilde{W}_{k-1}$. The change in W_k and V_k through added and/or deleted columns can analogously be applied to $widetildeW_k$. Example: insert new column at front. New columns: $(W_{k_n})_{i,0}$ and $(V_{k_n})_{i,0}$.

$$(\widetilde{W}_{k_n})_{i,0} = (W_{k_n})_{i,0} - \sum_{q=0}^{M} \widetilde{W}_{k_q} \left(V_{k_q}^{\dagger} \cdot (V_{k_n})_{i,0} \right) \tag{3}$$

Similar computation for insertion at arbitrary position. If column with index p is deleted from matrices W_k and V_k, delete also column with index p in $\widetilde{W}_k$.



- locally multiply: $V_{k_q}^{\dagger} \cdot (V_{k_n})_{i,0} =: \alpha \in \mathbb{R}^{k_q \times 1} \rightarrow$ `ALLREDUCE_SUM`
- fully locally multiply: $\widetilde{W}_{k_q} \cdot \alpha$ (embarrassingly parallel)

The matrix $\widetilde{W}_{k_n}$ needs to be re-computed if least-squares system is restored from backup (in case the previous time step converged within one iteration) or convergence is achieved for the current time step. In the latter case J_{prev}^{-1} is updated and hence the old entries in $\widetilde{W}_{k_n} = W_{k_n} - J_{prev}^{-1} V_{k_n}$ are outdated.

1.3 Compute (quasi-)Newton update Δx

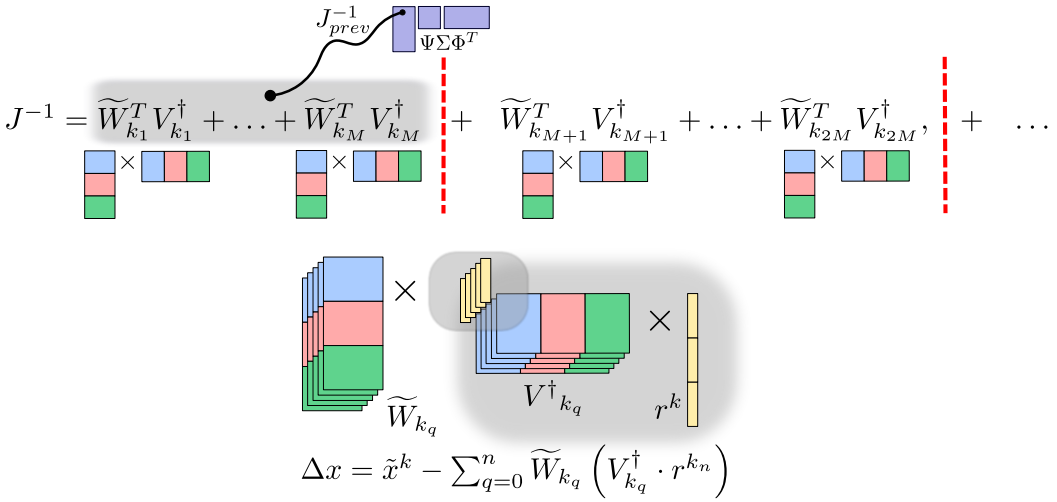
$$x^{k+1} = -\sum_{q=0}^{M} \widetilde{W}_{k_q} \left(V_{k_q}^{\dagger} r^k \right) - \widetilde{W}_{k_n} \left(V_{k_n}^{\dagger} r^k \right) \tag{4}$$

Similar to the update of $\widetilde{W}$, we need a `ALLREDUCE_SUM` operation and fully local multiplications of computational cost $\mathcal{O}(k_q \cdot {}^{N}/_{p})$ and $\mathcal{O}(k_q^2 \cdot {}^{N}/_{p})$, respectively. Hence total cost: $M \cdot \left(\mathcal{O}(k_q^2 {}^{N}/_{p}) + \mathcal{O}(\texttt{ALLREDUCE\_SUM}) \right)$

1.4 Restart-Mode for MVJ

The objective is to avoid the explicit computation and storage of the Jacobian matrix. At the end of each time step, the Jacobian is updated by $\widetilde{W}_{kn} V_{k_n}^{\dagger}$. Thus, the Jacobian can be written as

$$J^{-1} = \widetilde{W}_{k_1}^T V_{k_1}^{\dagger} + \widetilde{W}_{k_2}^T V_{k_2}^{\dagger} + \ldots + \widetilde{W}_{k_M}^T V_{k_M}^{\dagger}. \tag{5}$$



The matrices $\widetilde{W}_{k_q}$ and $V^\dagger_{k_n}$ are tall and skinny, thus the storage requirement as well as the computational costs for the computation of the (quasi-)Newton update are low for a small number of time steps. To maintain low complexity, a restart becomes necessary at some point. To that end, the simulation time is partitioned into chunks of M time steps. Within one chunk, the MVJ method computes a (quasi-)Newton update based on the minimization of the difference between two subsequent Jacobian approximations. However if it comes to restart at chunk borders, a suitable initial guess J_0^{-1} has to be found that should retain as much information as possible from the previous chunks.

We consider three different behaviours at restart:

RS-0. Clear all. This obviously results in $O(N \times K \times M)$ costs for both the storage of J^{-1} in the form (5) and for the M pairs of matrix-vector multiplications

$$y := V^\dagger_{k_q} x \quad \text{and} \quad \widetilde{W}_{k_q} y \quad \text{in} \quad J^{-1} x \quad \text{for any vector} \quad x \in \mathbb{R}^N.$$

RS-LS. Clear J^{-1}, but keep columns in V^{RS-LS} and W^{RS-LS} from time steps within the current chunk, i.e., use the initial guess $J^{-1} := 0$, $\widetilde{W}_0 := W$, and $V_0^\dagger := (V^T V)^{-1} V^T$. If we reuse at most $\bar{K}$ columns,the total costs are $O(N \times \bar{K}) + O(N \times K \times M)$.

- For the current implementation we only use the input/output information from the first `_usedColumnsPerTstep=5` iterations for the last `RSLSreusedTimesteps` time steps.
- The defined filter for the least-squares system is also applied to V^{RS-LS} and W^{RS-LS}.

RS-SVD. Do a subspace tracking based on a singular value decomposition (SVD) of the matrix J^{-1}

$$J^{-1} = \Psi\Sigma\Phi^T \quad \text{with} \quad \Sigma = diag(\sigma_1, \sigma_2, \ldots, \sigma_N)$$

where $\sigma_1 \geqslant \sigma_2 \geqslant \ldots \geqslant \sigma_N \geqslant 0$ are real singular values, and $\Psi \in \mathbb{R}^{N\times N}$ and $\Phi \in \mathbb{R}^{N\times N}$ are orthogonal matrices. At restart, we truncate this decomposition by cutting off all singular values below a given threshold, i.e., we restart with

$$J^{-1} = \underbrace{(\Psi_{\cdot,j})_{j=1,\ldots,\bar{K}}}_{=:\,\overline{\Psi}} \underbrace{\begin{pmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_{\bar{K}} \end{pmatrix}}_{=:\,\overline{\Sigma}} \underbrace{(\Phi_{\cdot,j})^T_{j=1,\ldots,\bar{K}}}_{=:\,\overline{\Phi}^T}. \tag{6}$$

The costs strongly depend on the efficient realization of the underlying SVD decomposition. Apart from this step, the total costs are $O(N \times \bar{K}) + O(N \times K \times M)$ if we truncate the SVD decomposition such that only $\bar{K}$ values are left. For the efficient implementation of the SVD, we assume that we have a truncated singular value decomposition as in (6). At the end of the next chunk, our new estimate reads

$$\overline{\Psi}\,\overline{\Sigma}\,\overline{\Phi}^T + \sum_{m=1}^{M} \widetilde{W}_{k_m} V^{\dagger}_{k_m} \tag{7}$$

for which we have to compute an updated truncated SVD by performing M low-rank updates of the form

$$\overline{\Psi}\,\overline{\Sigma}\,\overline{\Phi}^T + AB^T = \begin{bmatrix} \overline{\Psi} & A \end{bmatrix} \begin{bmatrix} \overline{\Sigma} & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} \overline{\Phi} & B \end{bmatrix}^T \tag{8}$$

with $A, B \in \mathbb{R}^{N\times k_m}$. We use the algorithm proposed in [**?**], i.e., we compute the orthogonal components of A and B. With the matrices Q_A and Q_B defining an orthonormal basis of the column space of $(I - \overline{\Psi}\,\overline{\Psi}^T)A$ and $(I - \overline{\Phi}\,\overline{\Phi}^T)B$, we define

$$R_A := Q_A^T(I - \overline{\Psi}\,\overline{\Psi}^T)A, \quad \text{and} \quad R_B := Q_B^T(I - \overline{\Phi}\,\overline{\Phi}^T)B.$$

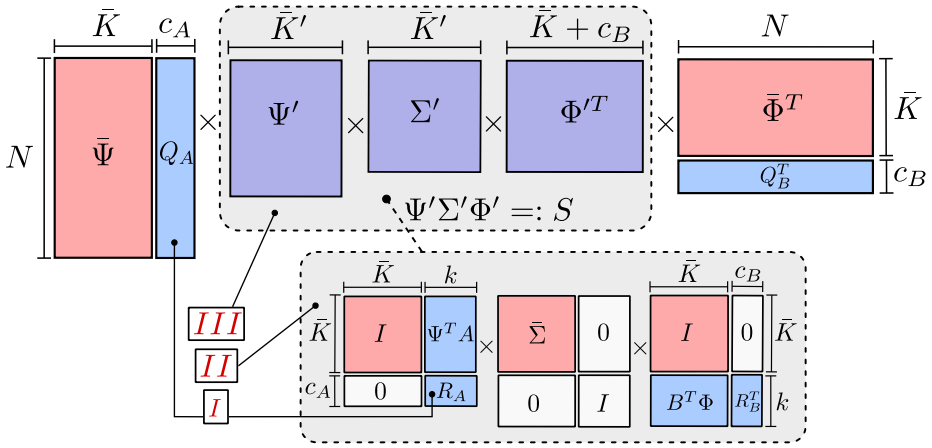
With this, the inverse Jacobian update (8) can be transformed to

$$\overline{\Psi}\,\overline{\Sigma}\,\overline{\Phi}^T + AB^T = \left[\overline{\Psi}\; Q_A\right] S \left[\overline{\Phi}\; Q_B\right]^T \text{ with } S = \begin{bmatrix} \overline{\Sigma} & 0 \\ 0 & 0 \end{bmatrix} + \underbrace{\begin{bmatrix} \overline{\Psi}^T A \\ R_A \end{bmatrix}}_{:=\,\mathcal{A}} \underbrace{\begin{bmatrix} \overline{\Phi}^T B \\ R_B \end{bmatrix}^T}_{:=\,\mathcal{B}}$$

Diagonalizing S as ${\Psi'}^T K \Phi' = \Sigma'$ finally yields

$$\overline{\Psi}\,\overline{\Sigma}\,\overline{\Phi}^T + AB^T = \left(\left[\overline{\Psi}\; Q_A\right] \Psi'\right) \Sigma' \left(\left[\overline{\Phi}\; Q_B\right] \Phi'\right)^T. \tag{9}$$

If $c_A,\ c_B,\ c := \max(c_A, c_B)$ is the dimension of the column space of $(I - \overline{\Psi}\,\overline{\Psi}^T)A$ and $(I - \overline{\Phi}\,\overline{\Phi}^T)B$, respectively, the costs for computing an orthonormal basis of these spaces are $O(c^2 N)$. The matrix $S \in \mathbb{R}^{(\bar{K}+c_A)\times(\bar{K}+c_B)}$ can be computed with $O(k_m \bar{K} N + c^2 N)$ (for the computation of $\mathcal{A}$ and $\mathcal{B}$) plus $O((\bar{K}+c)^2 k_m)$ (for the matrix-matrix multiplication $\mathcal{A}\mathcal{B}^T$) operations. The costs for the SVD of K depend only on the small number $\bar{K}+c$ and, thus, not on N. Summarizing, the costs for the update of the SVD are linear in N. After the update, the new SVD can be truncated again to keep a small but accurate representation.



1.4.1 RS-SVD Algorithm

(1) compute orthogonal components of A and B w.r.t. Φ and Ψ

$$\tilde{P} := (I - \Psi\Psi^T)A, \qquad Q_A := \text{orth}(\tilde{P}), \quad Q_A R_A = qr(\tilde{P}) \tag{10}$$

$$\tilde{Q} := (I - \Phi\Phi^T)B, \qquad Q_B := \text{orth}(\tilde{Q}), \quad Q_B R_B = qr(\tilde{P}) \tag{11}$$

$$R_A := Q_A^T(I - \Psi\Psi^T)A, \quad R_B := Q_B^T(I - \Phi\Phi^T)B$$

(2) build S with

$$S = \begin{bmatrix} I & \tilde{\Psi}^T A \\ 0 & R_A \end{bmatrix} \begin{bmatrix} S & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} I & \tilde{\Phi}^T B \\ 0 & R_B \end{bmatrix}^T = \begin{bmatrix} \overline{\Sigma} & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} \overline{\Psi}^T A \\ R_A \end{bmatrix} \begin{bmatrix} \overline{\Phi}^T B \\ R_B \end{bmatrix}^T$$

$$\overline{\Psi\Sigma\Phi}^T + AB^T = \begin{bmatrix} \overline{\Psi} \; Q_A \end{bmatrix} S \begin{bmatrix} \overline{\Phi} \; Q_B \end{bmatrix}^T$$

(3) diagonalizing S: compute SVD of S

$$S = \Psi'\Sigma'\Phi^{T'}$$

(4) rotate left and right eigenspaces:

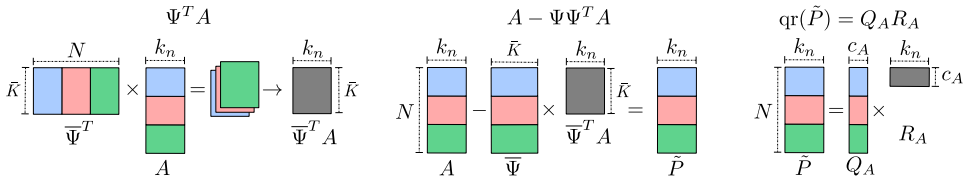
$$\overline{\Psi\Sigma\Phi}^T + AB^T = \underbrace{\left(\begin{bmatrix} \overline{\Psi} \; Q_A \end{bmatrix} \overline{\Psi}'\right)}_{\tilde{\Psi}} \underbrace{\overline{\Sigma}'}_{\tilde{\Sigma}} \underbrace{\left(\begin{bmatrix} \overline{\Phi} \; Q_B \end{bmatrix} \overline{\Phi}'\right)^T}_{\tilde{\Phi}^T}$$

(5) truncate SVD: Cut off if $\frac{\sigma_i}{\sigma_1} < \epsilon_{\text{trunc}}$

$$\overline{\Psi\Sigma\Phi}^T \leftarrow \tilde{\Psi}\tilde{\Sigma}\tilde{\Phi}^T$$

1.4.2 Parallel Implementation of SVD Update

(1) compute orthogonal component of A w.r.t. $\overline{\Psi} : (I - \Psi\Psi^T)A$:

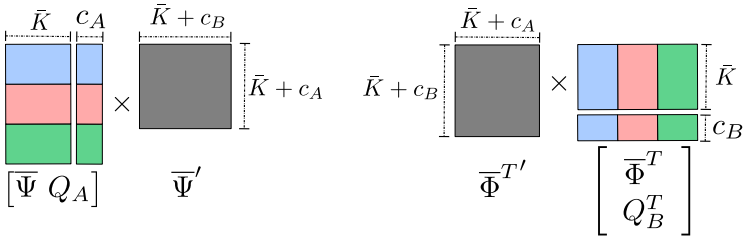


(2) compute orthogonal component of B w.r.t. $\overline{\Phi}$: $(I - \Phi\Phi^T)B$

(3) compute matrix S, all components are local, fully local multiplications.

(4) compute SVD of matrix S fully local: $S = \overline{\Psi}'\overline{\Sigma}'\overline{\Phi}'^T$

(5) rotate left and right subspace: $\overline{\Psi} \leftarrow \left[\overline{\Psi}\ Q_A\right]\overline{\Psi}'$ and $\left(\overline{\Phi} \leftarrow \left[\overline{\Phi}\ Q_B\right]\overline{\Phi}'\right)^T$



1.5 Preconditioning of Least-Squares System

Typically the coupling variables (pressure, displacement, ...) differ significantly in orders of magnitude. Therefore, the least-squares system needs to be scaled (preconditioned) properly if more than one coupling variable is accelerated by the quasi-Newton method, i. e., if the vectorial coupling scheme is applied. To maintain good conditioning of the least-squares system and the computation of the QR-factorization, the variables are supposed to be in the same order of magnitude.
Assumption for `preconditioner` interface: The scaling weights may change in each iteration for each entry.

Implemented variants:

a) `constant`: Scaling weights are defined in the `<data ... scaling="??"/>` tag for each coupling variable and never change during the simulation.

b) `residual`: Column-vector entries are scaled with the inverse of the residual entries (point-wise). Scaling weights change in each iteration but do not change in iterations converged. The QR-factorization needs to be re-computed in every iteration. Weights are potentially different for each entry.

c) `residual-sum`: Sub-vectors of column-vectors are scaled with the inverse L2-norm of the respective residual sub vectors. Scaling weights change in each iteration but do not change in iterations converged. The QR-factorization needs to be re-computed in every iteration. Weights are the same for one sub-vector.

d) `value`:

The Newton update reads

$$\Delta x = -J_{prev}^{-1} r^k - \left(\left(W_{k_n} - J_{prev}^{-1} V_{k_n} \right) \left(V_{k_n}^T V_{k_n} \right)^{-1} V_{k_n}^T \right) r^k \tag{12}$$

The least-squares system needs to be scaled with the `preconditioner`

$$\mathcal{P} = \begin{pmatrix} P_1 & & \\ & P_2 & \\ & & P_3 \end{pmatrix}, \quad \text{with } P_i = \operatorname{diag}(\omega_i^1, \ldots, \omega_i^n) \tag{13}$$

to maintain good conditioning for the QR-factorization. The canonical approach is to scale the entire update formula, i. e., all occurring vectors and matrices:

$$\begin{aligned}
\mathcal{P}\Delta x &= -\mathcal{P} J_{prev}^{-1} \mathcal{P}^{-1} r^k - \left(\left(\mathcal{P} W_{k_n} - \mathcal{P} J_{prev}^{-1} \mathcal{P}^{-1} \mathcal{P} V_{k_n} \right) \left(V_{k_n}^T \mathcal{P}^T \mathcal{P} V_{k_n} \right)^{-1} V_{k_n}^T \mathcal{P}^T \right) \mathcal{P} r^k \\
&= -\widetilde{J_{prev}^{-1}} \breve{r^k} - \left(\left(\widetilde{W}_{k_n} - \widetilde{J_{prev}^{-1}} \breve{V}_{k_n} \right) \left(\breve{V}_{k_n}^T \breve{V}_{k_n} \right)^{-1} \breve{V}_{k_n}^T \right) \breve{r^k} \\
&= -\mathcal{P} J_{prev}^{-1} r^k - \mathcal{P} \widetilde{W}_{k_n} \left(\mathcal{R}^{-1} \mathcal{Q}^T \mathcal{P}^{-1} \right) \mathcal{P} r^k \\
&= \mathcal{P} \left(-J_{prev}^{-1} r^k - \widetilde{W}_{k_n} \left(\breve{V^\dagger}_{k_n} \mathcal{P} \right) r^k \right) \\
\Rightarrow \Delta x &= -J_{prev}^{-1} r^k - \widetilde{W}_{k_n} (\underbrace{\breve{V^\dagger}_{k_n} \mathcal{P}}_{(\mathcal{R}^{-1} \mathcal{Q}^T \mathcal{P}^{-1}) \mathcal{P}}) r^k
\end{aligned}$$

With the scaled objects

$$\begin{aligned}
&\widetilde{J_{prev}^{-1}} := \mathcal{P} J_{prev}^{-1} \mathcal{P}^{-1}, \ \breve{W}_{k_n} := \mathcal{P} W_{k_n}, \ \breve{V}_{k_n} := \mathcal{P} V_{k_n}, \\
&\widetilde{\breve{W}}_{k_n} := \mathcal{P} \widetilde{W}_{k_n}, \ \breve{V^\dagger}_{k_n} := V^\dagger \mathcal{P}^{-1}, \ \breve{r^k} := \mathcal{P} r^k
\end{aligned}$$

However, from the above it is obvious that it suffices to scale $\breve{V}_{k_n} := \mathcal{P} V_{k_n}$ in order to compute the QR-factorization $\mathcal{P}\mathcal{Q}\mathcal{R} = \mathcal{P} V_{k_n}$ which finally results in the scaled pseudo inverse $\breve{V^\dagger}_{k_n} := V_{k_n}^\dagger \mathcal{P}^{-1}$. After the computation of the pseudo inverse it needs to be scaled back such that none of the other objects needs additional scaling $\breve{V^\dagger}_{k_n} \mathcal{P} = V_{k_n}^\dagger \mathcal{P}^{-1} \mathcal{P} = V_{k_n}^\dagger$.
This is correct as in particular

$$\| \widetilde{J_{prev}^{-1}} - \widetilde{J^{-1}} \| \rightarrow \min, \quad \text{s. t. } \widetilde{J_{prev}^{-1}} \breve{V}_{k_n} = \breve{W}_{k_n} \tag{14}$$

results in $\lambda^* = -\widetilde{\breve{W}}_{k_n} \breve{V^\dagger} = -\mathcal{P} \widetilde{W}_{k_n} V^\dagger \mathcal{P}^{-1} = -\left(\mathcal{P} W_{k_n} - \mathcal{P} J_{prev}^{-1} \mathcal{P}^{-1} \mathcal{P} V_{k_n} \right) V_{k_n}^\dagger \mathcal{P}^{-1}$ and

$$\| J_{prev}^{-1} - J^{-1} \| \rightarrow \min, \quad \text{s. t. } J_{prev}^{-1} V_{k_n} = W_{k_n} \tag{15}$$

results in $\lambda^* = -\widetilde{W}_{k_n} V^\dagger = \left(W - æ_{prev}^{-1} V_{k_n} \right) V^\dagger$. Thus, the scaling has no influence on the norm minimization condition.

Summarizing, the Jacobian update formula reads

$$J^{-1} = J_{prev}^{-1} + \widetilde{W}_{k_n} \breve{V^\dagger}_{k_n} \mathcal{P} \tag{16}$$

1.5.1 Preconditioning for MVJ-Restart Mode: SVD Update

For the computation of the SVD update we need to compute QR-factorizations of matrices that include column vectors of differences of coupling variables, i. e. $\widetilde{P} := (I - \breve{\Psi}\breve{\Psi}^T)\widetilde{\widetilde{W}}_{k_q}$ and $\widetilde{Q} := (I - \breve{\Phi}\breve{\Phi}^T)\widetilde{V^\dagger}_{k_q}$. Therefore, the update has to be computed from the scaled matrices $\widetilde{\widetilde{W}}_{k_q}$ and $\widetilde{V^\dagger}_{k_q}$ for $q = 1, \ldots, M$, i. e., we update a SVD of the scaled Jacobian matrix. This means, that for `preconditioners` that change their weights throughout the simulation, the weights need to be fixed to a certain value at the time of the first time step, i. e., after the first M time steps. However, experiments have shown, that the preconditioner weights do not change significantly after that first initialization phase of the simulation and therefore we observe similar quality in convergence speed.

When restart is triggered, the existing SVD needs to be updated with the scaled matrices, namely

$$\textbf{update:} \qquad \breve{\Psi}\Sigma\breve{\Phi} \leftarrow \breve{\overline{\Psi}}\Sigma\breve{\overline{\Phi}} + \sum_{q=0}^{M} \mathcal{P}\widetilde{W}_{k_q} V_{k_q}^\dagger \mathcal{P}^{-1} \tag{17}$$

$$\textbf{truncate:} \qquad \breve{\overline{\Psi}}\Sigma\breve{\overline{\Phi}} \leftarrow \breve{\Psi}\Sigma\breve{\Phi} \tag{18}$$

where $\breve{\Psi} := \mathcal{P}\Psi$ and $\breve{\Phi}^T := \left(\Phi\mathcal{P}^{-1}\right)^T$.

It is important that $\mathcal{P}^{(M)} = \mathcal{P}^{(k\cdot M)}$ for $k = 1, \ldots$, i. e., the preconditioner weights keep constant. Otherwise, the matrices Ψ and Φ are not unitary matrices any more and the singular value decomposition properties fall apart.

1.6 QR Decomposition and Re-orthogonalization

1.6.1 Updated QR Decomposition, Standard

We want to compute the factorization of $V = (v_1, v_2, \ldots, v_{k_n})$

$$QR = V \tag{19}$$

for $V \in \mathbb{R}^{N\times k_n}$, $Q \in \mathbb{R}^{N\times \tilde{k}_n}$ and $R \in \mathbb{R}^{\tilde{k}_n \times \tilde{k}_n}$, with $\tilde{k}_n \leqslant k_n$. The matrix $Q = (q1, \ldots, q_{k_n})$ is unitary and $Q^TQ = I$. We search for $q \in \mathbb{R}^N$, $r \in \mathbb{R}^{\tilde{k}_n}$ and $\rho \in \mathbb{R}$ such that:

$$(Q, v_i) = (Q, q)\begin{pmatrix} I & r \\ 0 & \rho \end{pmatrix} \quad \text{and} \quad Q^Tq = 0$$

Then $v_i = Qr + q\rho$ and $Q^Tv_i = r$. Let $v_i' := q\rho$, then:

$$v_i' = v - Qr = (I - QQ^T)v_i \qquad \text{(orthogonalization)}$$

Furthermore, $\|q\| \stackrel{!}{=} 1$, therefore

$$\rho = \|v_i'\| \;\text{ and }\; q := \frac{1}{\rho}v_i' \qquad \text{(normalization)}$$

The normalization step is not possible if $\rho = \|v_i'\| \approx 0$. In this case $v = Qr$ holds true and thus $v \in \text{span}\{q_1, q2, \ldots, Q_{\tilde{k}_n}\}$.

- **case** $k_n = N$: $\rightarrow q = 0, \rho = 0$.
- **case** $k_n < N$: choose arbitrary unit vector q that is orthogonal to Q. This keeps the orthogonality of Q. Q and V still have the same number of columns but the corresponding diagonal element in R is equal to zero $r_{ii} = 0$. This corresponds to deleting column i.
  Remark: The QR1 filter would delete this column, as $R(i,i) = r_{ii} = 0$. The QR2 filter would delete this column, as $\|v_i'\| / \|v_i\| < \epsilon_{QR2}$.

Re-orthogonalization: Due to round off errors and finite precision it is unlikely that $\rho = 0$ exactly, but $0 < \epsilon = \rho \ll 1$. Gram-Schmidt orthogonalization tries to make $\|Q^T v_i'\|$ small relative to $\|v_i\| \Rightarrow \|Q^T v_i'\| = \epsilon \|v_i\|$.
Suppose $\|Q^T v_i'\| = \epsilon |v_i\|$ is small for a small $\epsilon > 0$. Then, it holds

$$\|Q^T q\| = \epsilon \|v_i\| / \rho$$

for $q v_i' / \rho$. the error relative to $\|v_i\|$ can be arbitrarily large for a small ρ and we do not have necessarily $Q^T q = 0$ or nearly zero, as required.

To keep orthogonality of Q in the case when $\|v_i'\| / \|v_i\|$, we try to correct v_i' through a further Gram-Schmidt iteration applied to v_i':

$$s = Q^T v_i' \quad \text{and} \quad v_i'' = v_i' - Qs = v - Q(r + s)$$

and set $v_i' \leftarrow v_i''$ and $r \leftarrow r + s$. this process is repeated until $\|v_i''\| / \|v_i'\|$ is not too small. Implemented conditions for re-orthogonalization are

$$\frac{\|v_i'\|}{\|v_i\|} \leqslant 1/\theta = 0.7$$

which is similar to the QR2 filter with $\epsilon_{QR2} = 0.7$.

1.6.2 Updated QR Decomposition for SVD

When computing the QR-decomposition for $(I - \Psi\Psi^T)A$ and $(I - \Phi\Phi^T)B$, it is not possible to delete columns in the same way as we did for the QR-decomposition for the least-squares system. The QR-decomposition need so be modified correctly in this case. Compute

$$\tilde{P} = (I - \Psi\Psi^T)A \qquad \tilde{P} = Q_A R_A$$
$$\tilde{Q} = (I - \Phi\Phi^T)B \qquad \tilde{Q} = Q_B R_B$$

1. Orthogonalization:

$$Q_{A_{.,i}} = \tilde{P}_{.,i} - \sum_{j<i} r_{ij} Q_{A_{.,j}}$$
$$\Leftrightarrow \quad 1 \cdot Q_{A_{.,i}} + \sum_{j<i} r_{ij} \cdot Q_{A_{.,j}} = \tilde{P}_{.,i}$$

and $R_{A_{.,i}} = (r_{i,1}, r_{i,2}, \ldots, r_{i,i-1}, 1, 0, \ldots, 0)^T$.

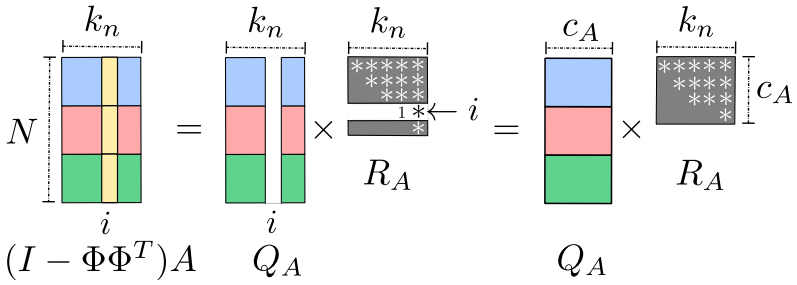


Figure 1: Eliminating a linear dependent column from the QR-decomposition for $(I - \Psi\Psi^T)A$

2. Normalization:

   $$Q_{A_{\cdot,i}} = {}^{Q_{A_{\cdot,i}}}/_{\|Q_{A_{\cdot,i}}\|}$$

   not possible if $\|Q_{A_{\cdot,i}}\| \approx 0$.

In this case the column is eliminated by deleting the corresponding column q_i from Q_A and deleting the corresponding row from R_A, i. e.,
In the least-squares case we delete the column from Q and also column and row from R. This corresponds to the deletion of one column of the least-squares system. However, this is not possible here, as $A \ni a_i \in \text{img}(\Psi)$ is not related to $\Psi^T A$ – this means we cannot delete the corresponding column in $\Psi^T A$.

Remark: If columns are eliminated as in Figure **??** we have to perform additional givens rotations to maintain a proper QR-decomposition. However, we do only eliminate columns at the end of the matrices, therefore only the last row of R_A is eliminated and givens rotations are not necessary.