APPENDIX B

Derivation of the (limited) factorised secant update scheme

B.1. Predictor-corrector scheme for the spurious drift term

The generalized S-QN equation is given by

$$\mathrm{d}\mathbf{x} = [-B(\mathbf{x})\nabla\Phi(\mathbf{x}) + k_B T\nabla\cdot B(\mathbf{x})]\mathrm{d}t + \sqrt{2k_BT}\,J(\mathbf{x})\mathrm{d}W(t). \tag{B-1}$$

We have previously shown [46] that (B-1) can be discretized using the predictor-corrector scheme introduced by Hütter and Öttinger [39] as

$$\begin{aligned}
\mathbf{x}_{k+1} &= \mathbf{x}_k + \Delta\mathbf{x}_k, && \text{(B-2)}\\
\Delta\mathbf{x}_k &= -\frac{1}{2}[B(\mathbf{x}_k+\Delta\mathbf{x}_k^p)\nabla\Phi(\mathbf{x}_k+\Delta\mathbf{x}_k^p) + B(\mathbf{x}_k)\nabla\Phi(\mathbf{x}_k)]\Delta t\\
&\quad + \frac{1}{2}[B(\mathbf{x}_k+\Delta\mathbf{x}_k^p)B^{-1}(\mathbf{x}_k) + I]\sqrt{2k_BT}\,J(\mathbf{x}_k)\Delta W_t, && \text{(B-3)}\\
\Delta\mathbf{x}_k^p &= -B(\mathbf{x}_k)\nabla\Phi(\mathbf{x}_k)\Delta t + \sqrt{2k_BT}\,J(\mathbf{x}_k)\Delta W_t. && \text{(B-4)}
\end{aligned}$$

where (B-4) is the predictor step and (B-2) is the correction. Direct inversion of B would be costly and should therefore be avoided. Using the Sherman-Morrison theorem, the exact inverse $G_k = G(\mathbf{x}_k) = B^{-1}(\mathbf{x}_k)$ of $B(\mathbf{x}_k)$ in dual space can be

calculated explicitly by

$$G_k = (I - \frac{\mathbf{y}_{k-1}\mathbf{s}_{k-1}^T}{\mathbf{y}_{k-1}^T\mathbf{s}_{k-1}})G_{k-1}(I - \frac{\mathbf{s}_{k-1}\mathbf{y}_{k-1}^T}{\mathbf{y}_{k-1}^T\mathbf{s}_{k-1}}) + \frac{\mathbf{y}_{k-1}\mathbf{y}_{k-1}^T}{\mathbf{y}_{k-1}^T\mathbf{s}_{k-1}}, \tag{B-5}$$

reusing the vectors $\mathbf{y}_{k-1}$ and $\mathbf{s}_{k-1}$ stored for updating B_{k-1}. Disregarding the costs associated with the computation of $\nabla\Phi(\mathbf{x}_k + \Delta\mathbf{x}_k^p)$ and the storage of G, we can calculate the costs of this predictor-corrector scheme employed for general Φ. For quadratic potentials, when the predictor (B-4) suffices and $\Delta\mathbf{x}_k = \Delta\mathbf{x}_k^p$, the total costs are $7n^2$ (see the theory section in Chapter 3). Due to the very related structure, the corrector equation (B-2) is $7n^2$ as well (if we reuse terms) plus an additional $2n^2$ for $B^{-1}(\mathbf{x}_k)$ using (B-5). The additional costs for (B-2) are thus $9n^2$ and the total costs for the predictor-corrector scheme using FSU are $16n^2$. The Sherman-Morrison theorem can also be applied to derive an analytic expression for $D_k = J_k^{-1}$ from (B-13), providing an efficient method for determining $B^{-1}(\mathbf{x}_k)$ for L-FSU. Again, the total costs of the full scheme are roughly doubled compared to using only the predictor term. Since this calculation is straightforward but involved, the full technical details are given in future publications for general Φ. As a concluding remark, we note that the calculation of the divergence itself may actually be more efficient than the predictor-corrector scheme, because of the special nature of the update $B(\mathbf{x}_k) = B(\mathbf{x}_{k-1}) + V$, with V a rank-two correction.

B.2. Derivation of the FSU algorithm

The derivation of the update for J is equivalent to the update for the lower triangular matrix L [33]. By interchanging $\mathbf{s}$ and $\mathbf{y}$ and replacing L with J, the matrices LL^T and JJ^T become approximates of the Hessian and the inverse Hessian respectively. Here we focus on the derivation of the update scheme for J.
Given $\|\cdot\|$ is the Frobenius norm and

$$\min_{J_{k+1}} \|J_{k+1} - J_k\|, \tag{B-6}$$

$$J_{k+1}\mathbf{v}_k = \mathbf{s}_k, \tag{B-7}$$

J_{k+1} is uniquely given by

$$J_{k+1} = J_k + \frac{\mathbf{s}_k - J_k\mathbf{v}_k}{\mathbf{v}_k^T\mathbf{v}_k}\mathbf{v}_k^T. \tag{B-8}$$

Substitute J_{k+1} into

$$J_{k+1}^T \mathbf{y}_k = \mathbf{v}_k, \tag{B-9}$$

gives

$$\begin{aligned} \mathbf{v}_k = J_{k+1}^T \mathbf{y}_k &= \left(J_k + \frac{\mathbf{s}_k - J_k \mathbf{v}_k}{\mathbf{v}_k^T \mathbf{v}_k} \mathbf{v}_k^T \right)^T \mathbf{y}_k, \\ &= J_k^T \mathbf{y}_k + \frac{(\mathbf{s}_k - J_k \mathbf{v}_k)^T \mathbf{y}_k}{\mathbf{v}_k^T \mathbf{v}_k} \mathbf{v}_k \qquad \text{(B-10)} \\ &\Rightarrow \\ (1 - \frac{(\mathbf{s}_k - J_k \mathbf{v}_k)^T \mathbf{y}_k}{\mathbf{v}_k^T \mathbf{v}_k}) \mathbf{v}_k &= J_k^T \mathbf{y}_k. \qquad \text{(B-11)} \end{aligned}$$

Hence, $\mathbf{v}_k = \alpha_k J_k^T \mathbf{y}_k$ and after substituting this into (B-10) gives

$$\alpha^2 = \frac{\mathbf{y}_k^T \mathbf{s}_k}{\mathbf{y}_k^T J_k J_k^T \mathbf{y}_k}, \tag{B-12}$$

which has a real solution for α due to the curvature condition and positive definiteness of $J_k J_k^T$. The update scheme for J_{k+1} is now given by

$$J_{k+1} = J_k + \frac{\alpha \mathbf{s}_k \mathbf{y}_k^T J_k - \alpha^2 J_k J_k^T \mathbf{y}_k \mathbf{y}_k^T J_k}{\mathbf{y}_k^T \mathbf{s}_k}. \tag{B-13}$$

Using this update we find after some algebraic operations that JJ^T is equal to the update derived from the BFGS scheme

$$\begin{aligned} J_{k+1} J_{k+1}^T &= J_k J_k^T - \frac{J_k J_k^T \mathbf{y}_k \mathbf{y}_k^T J_k J_k^T}{\mathbf{y}_k^T J_k J_k^T \mathbf{y}_k} + \frac{\mathbf{s}_k \mathbf{s}_k^T}{\mathbf{y}_k^T \mathbf{s}_k}, \\ &= B_k - \frac{B_k \mathbf{y}_k \mathbf{y}_k^T B_k}{\mathbf{y}_k^T B_k \mathbf{y}_k} + \frac{\mathbf{s}_k \mathbf{s}_k^T}{\mathbf{y}_k^T \mathbf{s}_k}. \qquad \text{(B-14)} \end{aligned}$$

B.3. The limited memory update scheme

We consider our L-FSU method in the framework of limited-memory approaches. To arrive at a limited-memory BFGS method, two different strategies have been used.

The L-BFGS method of Liu and Nocedal [64] recasts BFGS into a multiplicative form $B_{k+1} = V_k^T B_k V_k + \rho_k s_k s_k^T$, and truncates by only using the information stored in V_k and s_k during the last m updates. In particular, given a (often diagonal) B_0, the L-BFGS update is provided by

$$\begin{aligned} B_{k+1} &= (V_k^T ... V_{k-m+1}^T) B_0 (V_{k-m+1} ... V_k) \\ &\quad + \rho_{k-m+1} (V_k^T ... V_{k-m+2}^T) s_{k-m+1} s_{k-m+1}^T (V_{k-m+2} ... V_k) \\ &\quad + \rho_{k-m+2} (V_k^T ... V_{k-m+3}^T) s_{k-m+2} s_{k-m+2}^T (V_{k-m+3} ... V_k) \\ &\quad + \rho_{k-1} V_k^T s_{k-1} s_{k-1}^T V_k + \rho_k s_k s_k^T . \end{aligned} \tag{B-15}$$

This approach was recently generalized by Reed [71] for the convex Broyden family of Quasi-Newton updates. The variable storage conjugate gradient (VSCG) method of Buckley and LeNir [77] is based on the BFGS formula in the additive form and overwrites the most recent update once m is reached. If only the current update is stored, both algorithms reduce to the memoryless QN method of Shannon and Phua [78]. It is generally recognized that L-BFGS with Shanno scaling is the most efficient and reliable method across a range of test problems.
We can rewrite the update scheme for J_{k+1} in (B-13) as $J_{k+1} = V_k J_k = (\prod_{j=0}^{k} V_{k-j}) J_0$ with

$$V_k = (I - \frac{1}{\nu_k} v_k y_k^T), \tag{B-16}$$

with $v_k = h_k - s_k/\alpha_k$, $h_k = J_k J_k^T y_k$ and $\nu_k = h_k^T y_k$. Using the additional condition, $B_{k+1} = J_{k+1} J_{k+1}^T$, we obtain

$$B_{k+1} = J_{k+1} J_{k+1}^T = V_k V_{k-1} \quad ... \quad V_0 J_0 J_0^T V_0^T \quad ... \quad V_{k-1}^T V_k^T \tag{B-17}$$

$$= V_k J_k J_k^T V_k^T = V_k B_k V_k^T . \tag{B-18}$$

Rewriting this expression in the additive form, several terms cancel and we obtain exactly the additive Davidon-Fletcher-Powell (DFP) formula (see also appendix B.2) [71]. Hence, the multiplicative DFP formula

$$B_{k+1} = V_k^T B_k V_k + \rho_k s_k s_k^T \quad \text{with} \quad \mathrm{V_k} = (\mathrm{I} - \frac{1}{\nu_\mathrm{k}} \mathrm{y_k h_k^T}), \tag{B-19}$$

and the update scheme in FSU are equivalent. The principle difference is that we casted (B-19) into a *factorized* form (B-18). The recursive expression (B-17), obtained by loop unrolling, can serve as a basis for limited-memory implementation.

The recursive algorithm also allows for a limitation of the memory requirements of FSU, by storing at each k step vectors $\{y_k, s_k, h_k\}$ instead of matrices J_k and B_k in the original scheme (B-13), however, at the expense of an additional computational load. Since (B-13) is multiplicative, we adapt the L-BFGS strategy for limited-memory implementation of FSU (L-FSU). However, instead of truncating the incorporation of V_k in B, we truncate in J, i.e.

$$J_{k+1} = V_k V_{k-1} ... V_{k-m+1} J_0, \tag{B-20}$$

for $k \geq m$, and apply the second relation to update the mobility B

$$J_{k+1} J_{k+1}^T = V_k V_{k-1} \quad ... \quad V_{k-m+1} J_0 J_0^T V_{k-m+1}^T \quad ... \quad V_{k-1}^T V_k^T . \tag{B-21}$$

For $k < m$, the FSU relations apply. Upon comparing L-FSU to L-BFGS in (B-15), with V_k as in (B-19), we note three important properties: a) L-FSU is factorized, b) the memory requirements of L-FSU are the same as in L-BFGS, c) assuming $B_0 = I$, the number of matrix-vector products in L-FSU ($2m$) is of a different order than L-BFGS ($2m + m(m-1)$, or $2m + m(m-1)/2$ if case of re-using information). One remaining issue is whether the secant condition is satisfied by L-FSU for $k \geq m$.The L-Broyden family [71] was specially designed to satisfy the secant condition $B_{k+1} y_k = s_k$ for all k, since $V_k y_k = 0$. By construction, the L-FSU method satisfies the secant condition for $k < m$. Let $m > 1$ and $k \geq m$, we define a matrix $\tilde{B}_k = \tilde{J}_k \tilde{J}_k^T$ by

$$\tilde{J}_k = V_{k-1} ... V_{k-m+1} J_0, \tag{B-22}$$

and we find that

$$B_{k+1} y_k = J_{k+1} J_{k+1}^T y_k = \alpha_k (\tilde{h}_k - \beta_k h_k) + \beta_k s_k \ . \tag{B-23}$$

with $\tilde{h}_k = \tilde{B}_k y_k$ and $\beta_k = \tilde{h}_k^T y_k / h_k^T y_k$. Consequently, the secant condition is satisfied only when $\tilde{h}_k = h_k = J_k J_k^T y_k$, which is generally not the case. We now redefine V_k as

$$V_k = (I - \frac{1}{\tilde{h}_k^T y_k} (\tilde{h}_k - \frac{s_k}{\tilde{\alpha}_k}) y_k^T), \tag{B-24}$$

with $\tilde{\alpha}_k^2 = s_k^T y_k / \tilde{h}_k^T y_k$. Substituting this into (B-21) gives

$$J_{k+1} J_{k+1}^T \mathbf{y}_k = V_k \tilde{B}_k V_k^T \mathbf{y}_k = \tilde{\alpha}_k V_k \tilde{B}_k \mathbf{y}_k = \tilde{\alpha}_k V_k \tilde{\mathbf{h}}_k = \mathbf{s}_k, \tag{B-25}$$

and the secant condition is again satisfied. We note that only the h_k for $k \geq m$ are affected by this redefinition of V_k.

B.4. Recursive scheme for the limited memory update

The update scheme can be casted into
Algorithm 1.

$$\mathbf{d} = \mathbf{d}(\mathbf{x}_{K+1}); \tag{B-26}$$

$$\begin{cases} \textbf{for } i = K, ..., \max(0, K - m + 1) \\ \quad \mathbf{v}_i = \mathbf{h}_i - \mathbf{s}_i/\alpha_i; \\ \quad \lambda_i = \mathbf{v}_i^T \mathbf{d}; \\ \quad \mathbf{d} = \mathbf{d} - (\lambda_i/\mathbf{h}_i^T \mathbf{y}_i)\mathbf{y}_i; \\ \textbf{end} \end{cases} \tag{B-27}$$

$$\mathbf{d} = J_0 J_0^T \mathbf{d}; \tag{B-28}$$

$$\begin{cases} \textbf{for } i = \max(0, K - m + 1), ..., K \\ \quad \gamma_i = \mathbf{y}_i^T \mathbf{d}; \\ \quad \beta_i = \mathbf{y}_i^T \mathbf{d}; \\ \quad \mathbf{d} = \mathbf{d} - (\beta_i/\mathbf{h}_i^T \mathbf{y}_i)\mathbf{v}_i; \\ \textbf{end} \end{cases} \tag{B-29}$$

$$\textbf{stop with result } \mathbf{d} = J(\mathbf{x}_{K+1}) J(\mathbf{x}_{K+1})^T \mathbf{d}. \tag{B-30}$$

It is clear that for $K = k$, the procedure in Algorithm 1 provides the drift term in (3.6) for $\mathbf{d} = \mathbf{d}(\mathbf{x}_{k+1}) = -\nabla\Phi(\mathbf{x}_{k+1})\Delta t$ in (B-26). The noise term can be calculated using the second part of Algorithm 1, starting with (B-28) and $\mathbf{d} = \sqrt{2k_B T}\, J_0 \Delta W_t$. For $k < m$, the vector $\mathbf{h}_k = J_k J_k^T \mathbf{y}_k$ can also be obtained using Algorithm 1 by setting $\mathbf{d} = \mathbf{y}_k$ and $K = k - 1$. Consequently, we obtain α_k from

$$\alpha_k = \alpha_k(\mathbf{h}_k) = \sqrt{\frac{\mathbf{s}_k^T \mathbf{y}_k}{\mathbf{h}_k^T \mathbf{y}_k}}, \tag{B-31}$$

and store this new value α_k in a vector α. For $k \geq m$, $\tilde{\mathbf{h}}_k = \tilde{B}_k \mathbf{y}_k$ can be obtained from Algorithm 1 starting with $\mathbf{d} = \mathbf{y}_k$ with the recursive index running between $k-1$

to $k - m + 1$. We store $\alpha_k = \alpha_k(\tilde{\mathbf{h}}_k)$ and $\mathbf{h}_k = \tilde{\mathbf{h}}_k = \mathbf{d}$. This scheme requires only permanent storage of vector-triplets $\{\mathbf{s}_k, \mathbf{y}_k, \mathbf{h}_k\}$ (each of length n) for each iteration step k. In agreement with general practice the small additional effort for storing and calculating the vector α of length m is not considered in the analysis [21].
Upon analysing the computational load, operations (B-27) and (B-29) add up to $3mn$ and $2mn$ multiplications, respectively. An additional n operations are needed for (B-28), if we assume J_0 is a diagonal (positive definite) matrix, giving rise to $5mn + n$ operations. Recursive calculation of $\mathbf{h}_k$ requires a maximum of $5mn + n$ operations (for $k = m - 1$), and slightly less for other k. The total is a maximum of $10mn + 2n$ multiplications per step for the drift term only. For the noise term only the second part of the algorithm is required. Assuming again a diagonal J_0, we find that n multiplications are required for $\sqrt{2k_BT}J_0\Delta W_t$ and $2mn$ multiplications for (B-29). This brings us to a total of $2mn + n$ multiplications for the noise term, and a total of $12nm + 3n$ for the complete cycle at time step k.