Calculating Propagators and Saving Memory

For continuous-Gaussian-chain models such as the "standard" model used in PSCF, the architecture of an ***acyclic*** block copolymer (BCP) chain, which consists of $n_B \geq 1$ (linear homopolymer) **blocks** connecting $n_V = n_B + 1$ **vertices** (note that an acyclic chain can be constructed by adding, in each step, a block and a new vertex to an existing vertex), including both joints (J) and free ends (E), is described by specifying the vertices j and $k \in [0, n_B]$ connected by each block i (=0,…,n_B−1). Each block i has two (one-end-integrated) propagators $q_i(\mathbf{r},s)$ and $q_i^\dagger(\mathbf{r},s)$, where $\mathbf{r}$ denotes the spatial position, and s=0,…,$N_i\Delta s$ with N_i being the number of steps along the chain contour into which the block is uniformly discretized, $\Delta s \equiv f_i / N_i$ being the step-size, and f_i being the volume fraction of block i in the copolymer chain; in the REPS-K method (K=0,…,4; see REPS.pdf for details), N_i must be an integer multiple of 2^K (*i.e.*, $M_i \equiv N_i / 2^K$ must be a positive integer). PSCF calculates the propagators in two steps: in Step I it solves all the forward propagators $\{q_i(\mathbf{r},s)\}$, and in Step II it solves all the backward propagators $\{q_i^\dagger(\mathbf{r},s)\}$; that is, for each BCP component in the system, the labeling of the two end-segments s=0 and $s=f_i$ for all its blocks and the order of solving all its propagators are determined automatically in PSCF based on its chain architecture.

Hereafter we take the calculation of propagators for block i as an example. To calculate its contribution to the volume-fraction field of its segment type, we need to evaluate $\int_0^{f_i} \mathrm{d}s q_i(\mathbf{r},s) q_i^\dagger(\mathbf{r},s) \approx \Delta s \sum_{k'=1}^{M_i} \sum_{k=0}^{2^K} c_k q_{i,2^K(k'-1)+k} q^\dagger_{i,2^K(k'-1)+k}$ at all $\mathbf{r}$ using the Romberg integration (denoted by RI-K; see RI.pdf for details), where c_k's are the coefficients used in RI-K, $q_{i,j} \equiv q_i(\mathbf{r}, j\Delta s)$, and $q^\dagger_{i,j} \equiv q_i^\dagger(\mathbf{r}, j\Delta s)$. Usually (as in PSCF) such integrals are calculated after all the propagators are solved and stored. The "slice" algorithm proposed in Ref. 1, however, is based on the fact that once the contribution of segment s on block i to the above integral (and stresses) is calculated, $q_{i,j}$ and $q^\dagger_{i,j}$ are no longer needed. In particular, $q_{i,j}$ for all i are stored only at their "**check points**", where j=0 or $j = N_i - \left(\sum_{i=1}^{l} i - 1\right) = N_i - \frac{l(l+1)}{2} + 1$ for ***integer*** values of $l \in [1, l_{\max})$ (in the ***descending*** order of l), in Step I above; to ensure j>0 in the latter case, one finds $l_{\max} = \left(\sqrt{8N_i + 9} - 1\right)/2$. This splits block i into (at least two) "**slices**", each containing one "check point" corresponding to the smallest j-value in the "slice". In Step II above, only $q^\dagger_{i,j^*}$ for the current value of j^* (=N_i,N_i−1,…,0) is stored, and the above integral is calculated (accumulated) "slice" by "slice"; note that $q^\dagger_{i,j^*-1}$ needs to be calculated out-of-place in REPS-K due to its successive halving of Δs. In the "slice" containing j^*, if q_{i,j^*} is available (*e.g.*, when j^* is a check point), the contribution of j^* to the above integral is then directly calculated; otherwise, all $q_{i,j}$ from the "check point" in the "slice" are re-calculated and stored in the same places as the no-longer-needed q-values, then their contribution to the above integral is calculated.

The procedure of the "slice" algorithm can be illustrated with N_i=8 as follows:

(1) $q_{i,j=0}$ is the ***known*** initial condition of the forward propagator. Since j=0 is a "check point", $q_{i,j=0}$ is stored in `q[0]`, the first element of a ***four-element array*** for storing the forward

propagators (note that each element is actually an array for storing the propagator values at all **r**). $q_{i,j}$ for j=1,2,…,8 are then obtained ***sequentially*** by solving the modified diffusion equation (MDE) for the forward propagator via, for example, the REPS-3 method; since here only j=3, 6 and 8 (*i.e.*, l=3, 2 and 1, respectively) are "check points", $q_{i,j=3}$, $q_{i,j=6}$ and $q_{i,j=8}$ are stored in `q[1]`, `q[2]` and `q[3]`, respectively. Therefore, there are four "slices".

(2) At the end of Step I, all the forward propagators are calculated, which then gives the normalized single-chain partition function of each component and $q^{\dagger}_{i,j=N_i}$ (*i.e.*, the initial condition of the backward propagator). Since $j=N_i$ (*i.e.*, l=1) is always a "check point" (which is the only j-value contained in "Slice" 1), $q^{\dagger}_{i,j=N_i}$ is stored as `qd`, the ***variable*** for storing the backward propagators (again it is actually an array for storing the propagator values at all **r**), and the contribution of segment $s=f_i$ to the above integral, $c_8 q_{i,j=8} q^{\dagger}_{i,j=8}$, is calculated using `q[3]` and `qd`.

(3) "Slice" 2 contains j=7 and 6. $q^{\dagger}_{i,j=7}$ is obtained from `qd` by solving the MDE for the backward propagator and stored in `qd` (which is overwritten since $q^{\dagger}_{i,j=8}$ is no longer needed). Since j=7 is not a "check point", $q_{i,j=7}$ is calculated (again) from `q[2]` by solving the MDE for the forward propagator and stored in `q[3]`. The contribution $c_7 q_{i,j=7} q^{\dagger}_{i,j=7}$ to the above integral is then calculated (and accumulated) using `q[3]` and `qd`. Similar to $q^{\dagger}_{i,j=7}$, $q^{\dagger}_{i,j=6}$ is obtained from `qd` and then stored in `qd`. Since j=6 is a "check point", however, the contribution $c_6 q_{i,j=6} q^{\dagger}_{i,j=6}$ to the above integral is calculated (and accumulated) using `q[2]` and `qd`.

(4) "Slice" 3 contains j=5, 4 and 3. Similar to (3), $q_{i,j=4}$ is calculated (again) from `q[1]` and stored in `q[2]`, and $q_{i,j=5}$ is then calculated (again) from `q[2]` and stored in `q[3]`; this clever design, where the number of j-values contained in the "slices" follow an arithmetic sequence of 1,2,3,…, efficiently re-uses the no-longer-needed storage for the forward propagators. For j=5, 4 and 3, $q^{\dagger}_{i,j}$ is obtained from `qd` and then stored in `qd`, and the contribution $c_j q_{i,j} q^{\dagger}_{i,j}$ to the above integral is then calculated (and accumulated) using `q[3]`, `q[2]` and `q[1]`, respectively, and `qd`.

(5) The last "slice", "Slice" 4, contains the rest j-values (*i.e.*, j=2, 1 and 0). Similar to (4), $q_{i,j=1}$ is calculated (again) from `q[0]` and stored in `q[1]`, and $q_{i,j=2}$ is then calculated (again) from `q[1]` and stored in `q[2]`. For j=2, 1 and 0, $q^{\dagger}_{i,j}$ is obtained from `qd` and then stored in `qd`, and the contribution $c_j q_{i,j} q^{\dagger}_{i,j}$ to the above integral is then calculated (and accumulated) using `q[2]`, `q[1]` and `q[0]`, respectively, and `qd`.

Compared to the usual approach of calculating and storing all propagators ***before*** calculating the volume-fraction fields, the "slice" algorithm reduces the memory usage by a factor of $\sqrt{2N_i}$ for large N_i at the cost of increasing the computation by 50% (*i.e.*, solving $q_{i,j}$ twice). Similarly, to calculate the contribution of block i to the system stresses[2] (which vanish when the bulk periodicity of an ordered phase formed by BCP self-assembly is found), we evaluate $\int_0^{f_i} \mathrm{d}s \hat{q}_i(\mathbf{q},s)\hat{q}^{\dagger}_i(-\mathbf{q},s)$ in the same way as above, where $\hat{q}_i(\mathbf{q},s) \equiv \int \mathrm{d}\mathbf{r} \exp\left(-\sqrt{-1}\mathbf{q}\bullet\mathbf{r}\right) q_i(\mathbf{r},s) \Big/ V$ and $\hat{q}^{\dagger}_i(\mathbf{q},s) \equiv \int \mathrm{d}\mathbf{r} \exp\left(-\sqrt{-1}\mathbf{q}\bullet\mathbf{r}\right) q^{\dagger}_i(\mathbf{r},s) \Big/ V$ with $\mathbf{q}$ being the wavevector and V the system

volume.

Finally, a slightly different “slice” algorithm can be used for discrete-chain models and will be implemented in PSCF+ soon.

References:

1. Qiang, Y. Chap. 6.3.2 in The High-Performance Algorithms for Self-Consistent Field Theory. PhD Thesis, Fudan University, Shanghai, China, 2022.
2. Tyler, C. A.; Morse, D. C., Stress in Self-Consistent-Field Theory. *Macromolecules* **2003,** *36* (21), 8184-8188.