Some Tips for Using CKTSO in Circuit Simulators

1. Selection of `CKTSO_Factorize` and `CKTSO_Refactorize`

In DC simulation, `CKTSO_Factorize` is always recommended.

In transient simulation, the following method is recommended. Conventional SPICE-style simulators use the following method to judge whether Newton-Raphson iterations are **converged**

$$\left|\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}\right| < \varepsilon_a + \varepsilon_r \times \min\{||\boldsymbol{x}^{(k)}||, ||\boldsymbol{x}^{(k-1)}||\}$$

where ε_a and ε_r are given absolute and relative thresholds, respectively. One can simply use the same method **with larger thresholds** to judge whether Newton-Raphson iterations are **converging**. If so, it means that the matrix values are changing slowly and `CKTSO_Refactorize` can be used; otherwise `CKTSO_Factorize` should be used. The thresholds for judging whether Newton-Raphson iterations are converging may be empirically determined. Note that the first factorization must call `CKTSO_Factorize`.

2. Fast factorization of `CKTSO_Factorize`

The last argument of `CKTSO_Factorize` specifies whether to enable fast factorization. Please note that even if fast factorization is enabled, the first factorization is much slower than subsequent factorizations.

In DC simulation, there are some independent simulation "methods", like direct Newton-Raphson iterations, GMIN stepping, source stepping, and pseudo-transient. The first factorization of each "method" should disable fast factorization. To be more conservative, for each different stepping value in stepping-based methods, the first factorization may also disable fast factorization.

In transient simulation, generally fast factorization can always be used, with the exception that each time after the time node is rolled back due to divergence, the next factorization should also disable fast factorization.

In both DC and transient simulations, once factorization fails due to numerical singularity, the next factorization should disable fast factorization.

3. Forced sequential solving of `CKTSO_Solve`

The last argument of `CKTSO_Solve` specifies whether to force CKTSO to use sequential solving. Please note that parallel solving requires an initialization step that spends about 2-5X of sequential solving time. The initialization is executed every time after the symbolic structure of the LU factors is changed (re-pivoting happens).

First of all, parallel solving have different performance for different matrix storage orders and on different systems. For column-mode matrices on Windows, parallel solving does not tend to get better performance so it is not recommended. For other situations parallel solving may get performance benefits, and whether to use forced sequential solving can be determined as follows.

In DC simulation, forced sequential solving is recommended.

In transient simulation, forced sequential solving should be set for `CKTSO_Factorize`. For `CKTSO_Refactorize`, solving can be parallel. Please note that even if forced sequential solving is disabled, CKTSO may still use sequential solving, according to some internal strategies.

4. Ax values of `CKTSO_Analyze`

`CKTSO_Analyze` is usually a one-time work. The matrix ordering and static pivoting are both done based on the symbolic pattern and values of the matrix which are provided to `CKTSO_Analyze`. In circuit simulation iterations, the matrix values are changing. Hence, the values provided to `CKTSO_Analyze` should be representative and have similar features of the matrix values during the iterations of an entire simulation process. This can be done by adding a specialized value filling function which fills typical matrix values. In this process, if a value is expected to frequently become 0 in circuit simulation iterations, it should be filled with 0.