

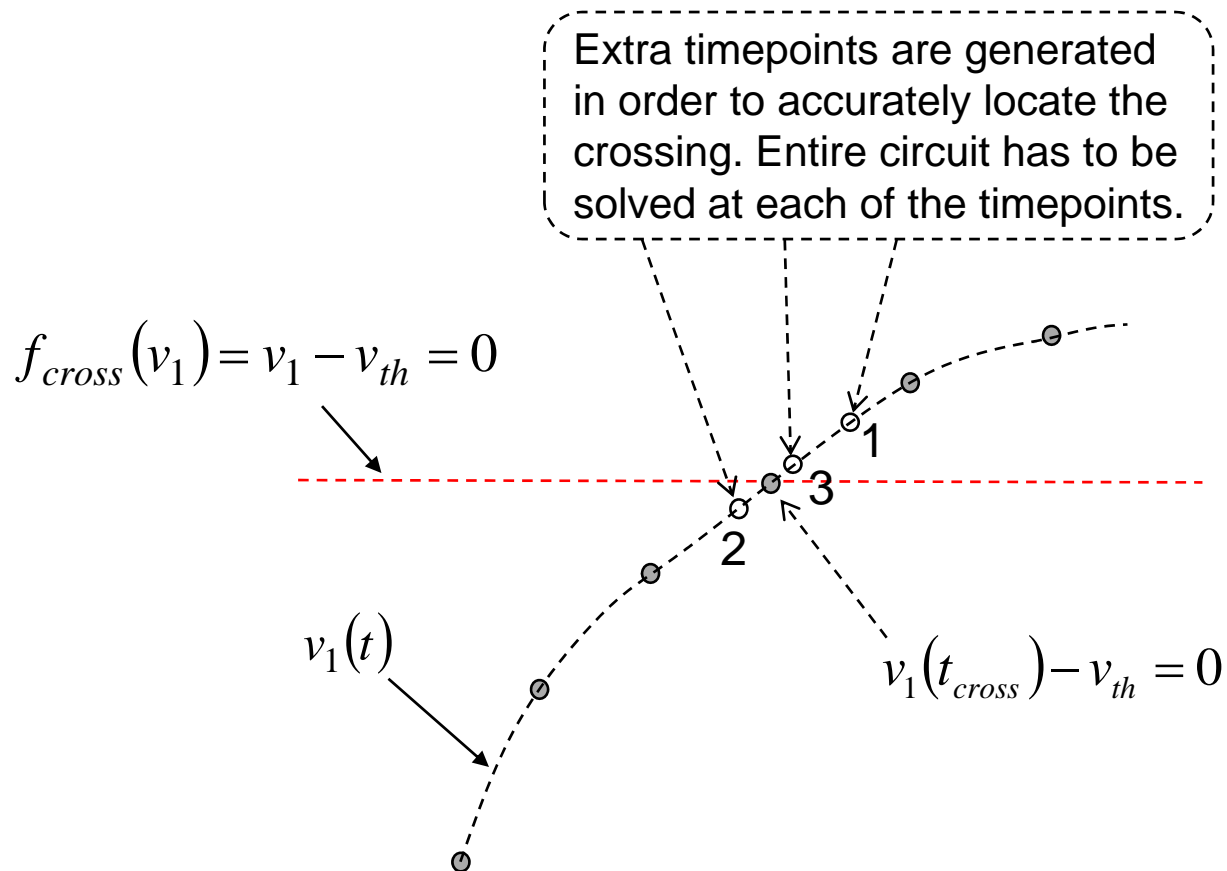
An Efficient Method to Simulate Threshold Crossing Events

G. Peter Fang
Texas Instruments

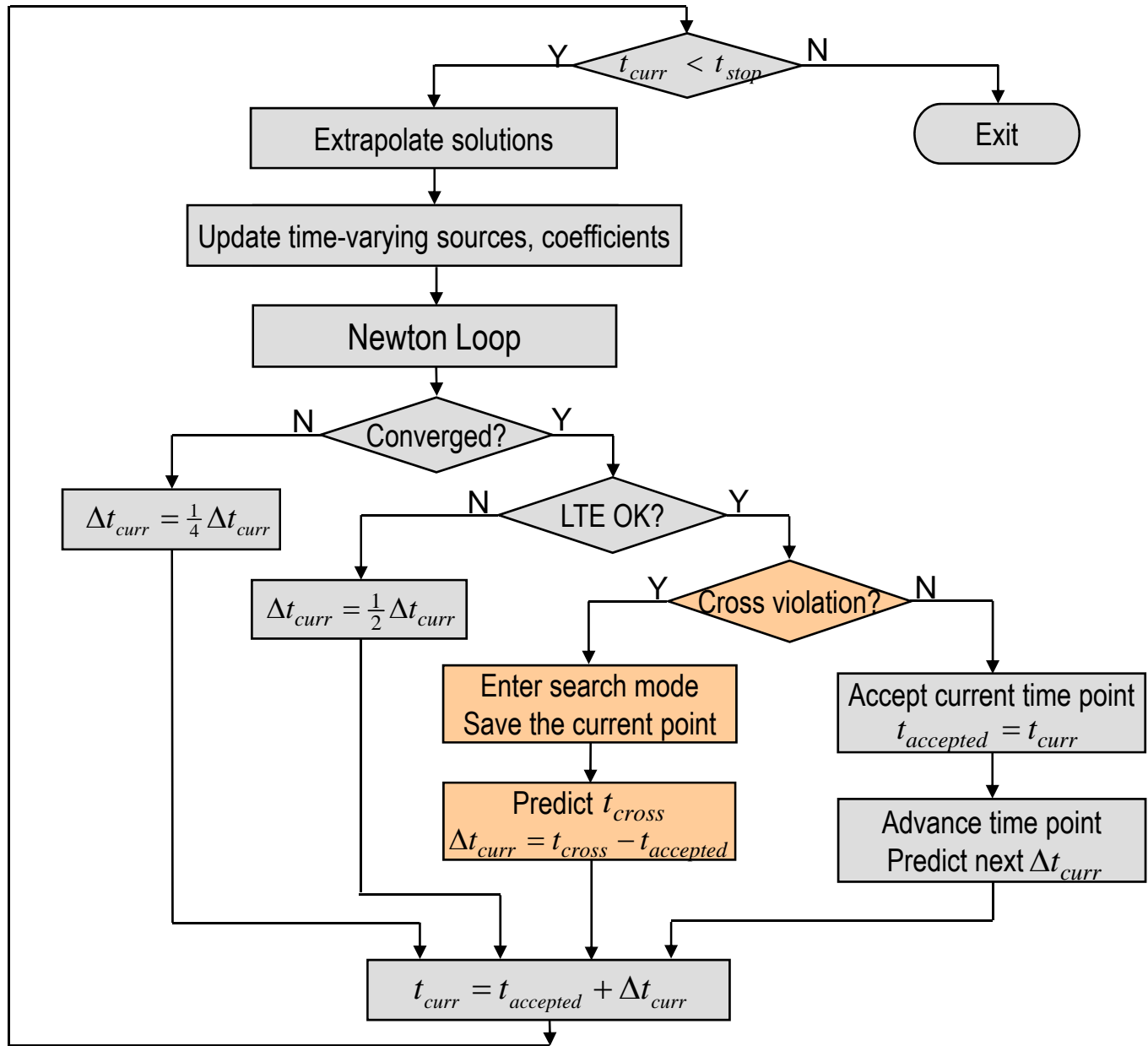
Threshold-Crossing

- Cross event has been widely used in behavioral simulation to change the dynamics of the model when a certain analog expression crosses through zero.
- The cross() function in Verilog-A generates a monitored analog event to detect threshold crossing; It controls the timestep to accurately resolve the crossing.
- The cross() function is heavily used in A2D events in Verilog-AMS for threshold crossing.
- **The Problem:** The use of the @cross event could force an excessive number of timepoints being evaluated, thus slows down the simulation significantly.

The Existing Method



Existing Transient Flow



The New Method

- When crossing, treat the timestep as an unknown and solve circuit equations and the cross equations together as one nonlinear system
- Directly converge to the solution at the crossing
 - No extra timepoints needed

The Coupled Nonlinear System

$$\bar{f}_{ckt}(\bar{v}_m) = \frac{q(\bar{v}_m) - q(\bar{v}_{m-1})}{h_m} + i(\bar{v}_m) + \bar{u}_m = 0$$

$$f_{cross}(\bar{v}_m) = 0$$



Treat the timestep as an independent variable
 $N+1$ equations, $N+1$ unknowns

$$\bar{F}(\bar{v}_m, h_m) = 0$$

Newton Method

$$\begin{bmatrix} \bar{J}_{ckt} \equiv \frac{\partial \bar{f}_{ckt}}{\partial \bar{v}_m} & \frac{\partial \bar{f}_{ckt}}{\partial h_m} \\ \frac{\partial f_{cross}}{\partial \bar{v}_m} & 0 \end{bmatrix} \begin{bmatrix} \Delta \bar{v}_m^{k+1} \\ \Delta h_m^{k+1} \end{bmatrix} = \begin{bmatrix} -\bar{f}_{ckt} \\ -f_{cross} \end{bmatrix}$$

$$\bar{v}_m^{k+1} = \bar{v}_m^k + \Delta \bar{v}_m^{k+1}$$

$$h_m^{k+1} = h_m^k + \Delta h_m^{k+1}$$

Update the timestep during the iteration!

Solving the Linear System I

Multiply both sides with

$$\begin{bmatrix} \overline{-1} & 0 \\ \overline{J}_{ckt} & 1 \\ 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \overline{I}_{N \times N} & \overline{J}_{ckt}^{-1} \frac{\partial \overline{f}_{ckt}}{\partial h_m} \\ \frac{\partial f_{cross}}{\partial \overline{v}_m} & 0 \end{bmatrix} \begin{bmatrix} \Delta \overline{v}_m^{k+1} \\ \Delta h_m^{k+1} \end{bmatrix} = \begin{bmatrix} \overline{-1} \overline{J}_{ckt} \overline{f}_{ckt} \\ -f_{cross} \end{bmatrix}$$

Solving the Linear System II

Perform Gauss Elimination

$$\begin{bmatrix} \bar{I}_{N \times N} & \bar{J}_{ckt}^{-1} \frac{\partial \bar{f}_{ckt}}{\partial h_m} \\ -\frac{\partial f_{cross}}{\partial \bar{v}_m} \cdot \left(\bar{J}_{ckt}^{-1} \frac{\partial \bar{f}_{ckt}}{\partial h_m} \right) & \Delta \bar{h}_m^{k+1} \end{bmatrix} \begin{bmatrix} \Delta \bar{v}_m^{k+1} \\ \Delta \bar{h}_m^{k+1} \end{bmatrix} = \begin{bmatrix} -\bar{J}_{ckt}^{-1} \bar{f}_{ckt} \\ f_{cross} + \frac{\partial f_{cross}}{\partial \bar{v}_m} \cdot \left(\bar{J}_{ckt}^{-1} \bar{f}_{ckt} \right) \end{bmatrix}$$

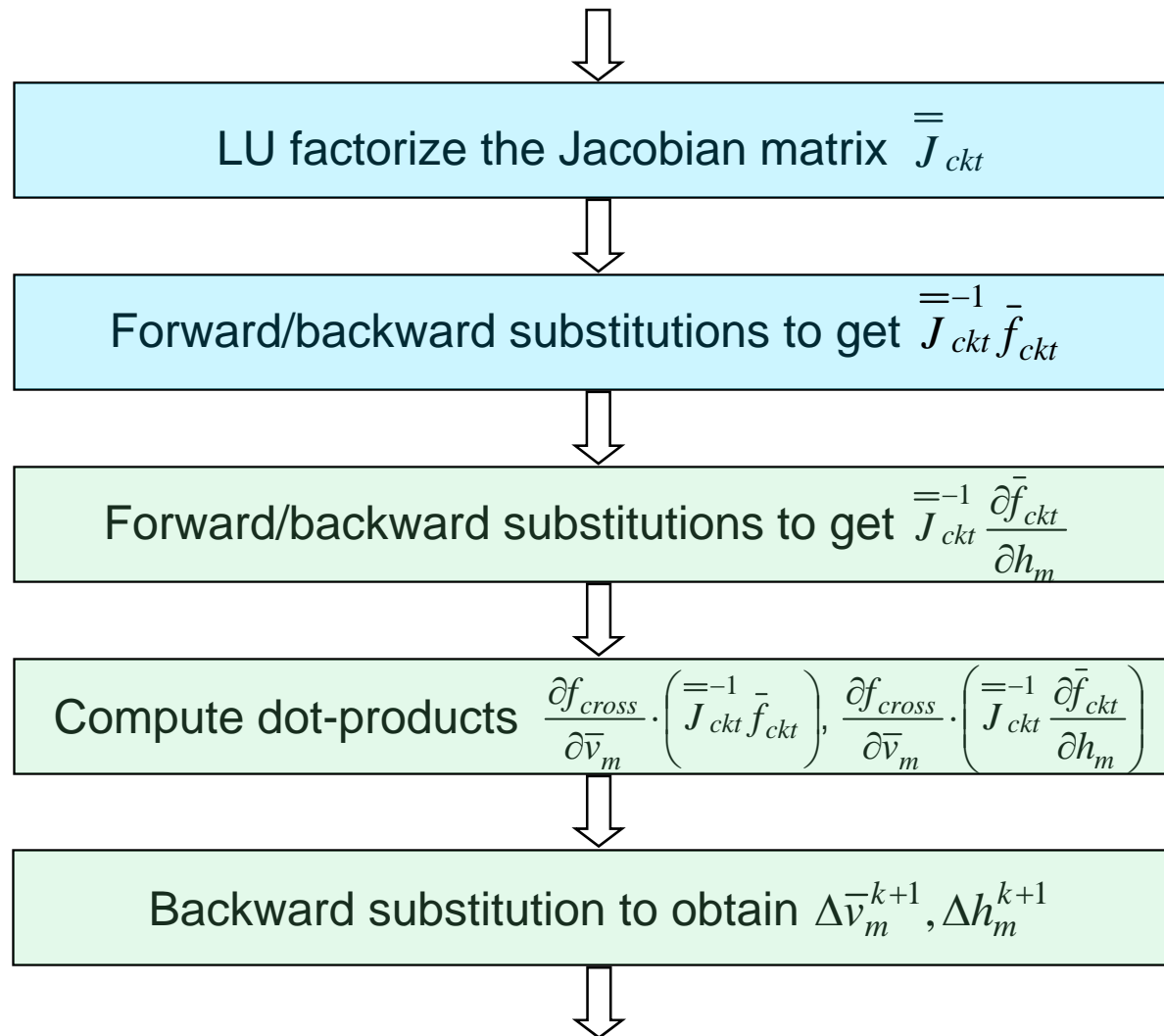
Solving the Linear System III

Backward substitution to obtain the solution vector

$$\Delta h_m^{k+1} = \frac{f_{cross} + \frac{\partial f_{cross}}{\partial \bar{v}_m} \cdot \left(\bar{J}_{ckt}^{-1} \left(-\bar{f}_{ckt} \right) \right)}{\frac{\partial f_{cross}}{\partial \bar{v}_m} \cdot \left(\bar{J}_{ckt}^{-1} \frac{\partial \bar{f}_{ckt}}{\partial h_m} \right)}$$

$$\Delta \bar{v}_m^{k+1} = \bar{J}_{ckt}^{-1} \left(-\bar{f}_{ckt} \right) + \left(\bar{J}_{ckt}^{-1} \frac{\partial \bar{f}_{ckt}}{\partial h_m} \right) \cdot \Delta h_m^{k+1}$$

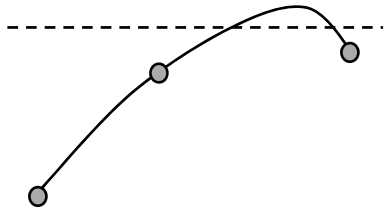
Flow for the Linear Solver



Implementation Details

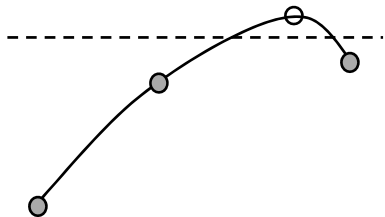
- Improved cross violation detection
- Evaluate/check all cross conditions at the beginning of each Newton iteration
 - Enter the search mode and re-compute timestep if a violation is detected
 - In search mode, extrapolate, update time-varying sources and time coefficients
- Switch to the coupled solver in search mode
 - Treat the timestep as an unknown
 - Solve the circuit equations and the cross condition together as one nonlinear system
 - Adjust the timestep during the Newton iteration

Improved Cross Violation Detection



$$f_{cross}(t_{m-1}) \cdot f_{cross}(t_{m-1} + h_m) < 0?$$

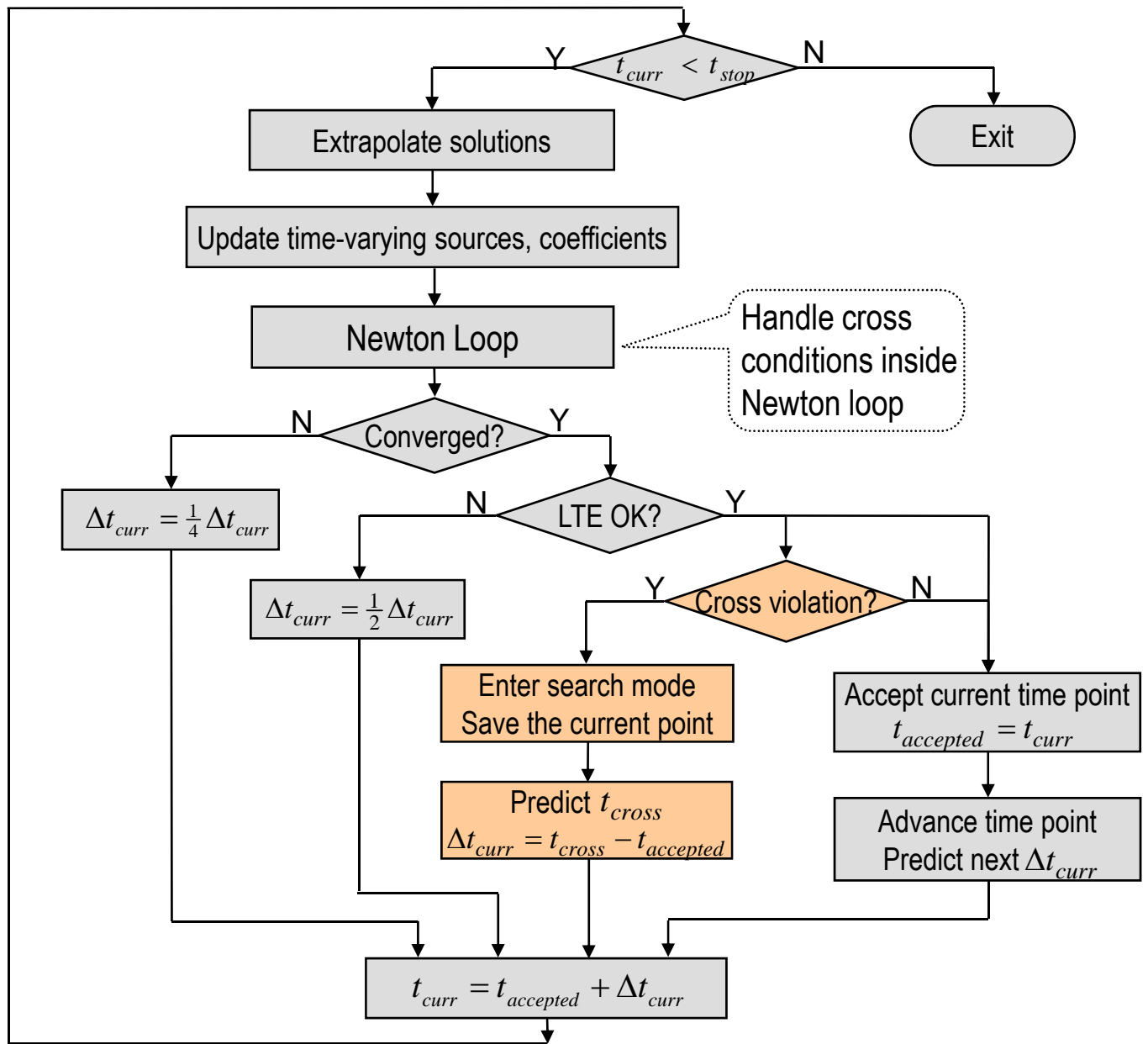
The existing method could miss the crossing



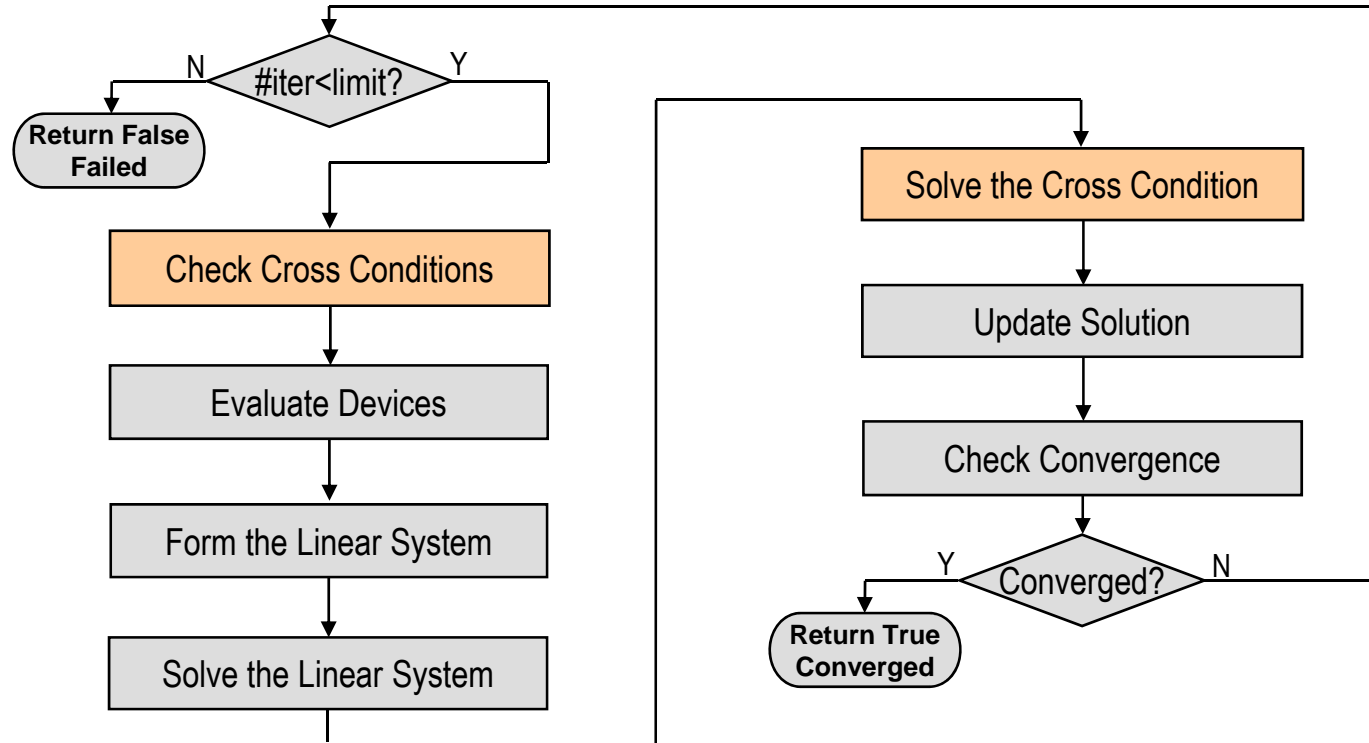
$$f_{cross}(t_{m-1}) \cdot f_{cross, \max/\min} < 0?$$

The new method utilizes interpolated 2nd order polynomial to find local maximum or minimum between current and previous timepoints then detect the crossing using that value. This method is one order of magnitude more accurate.

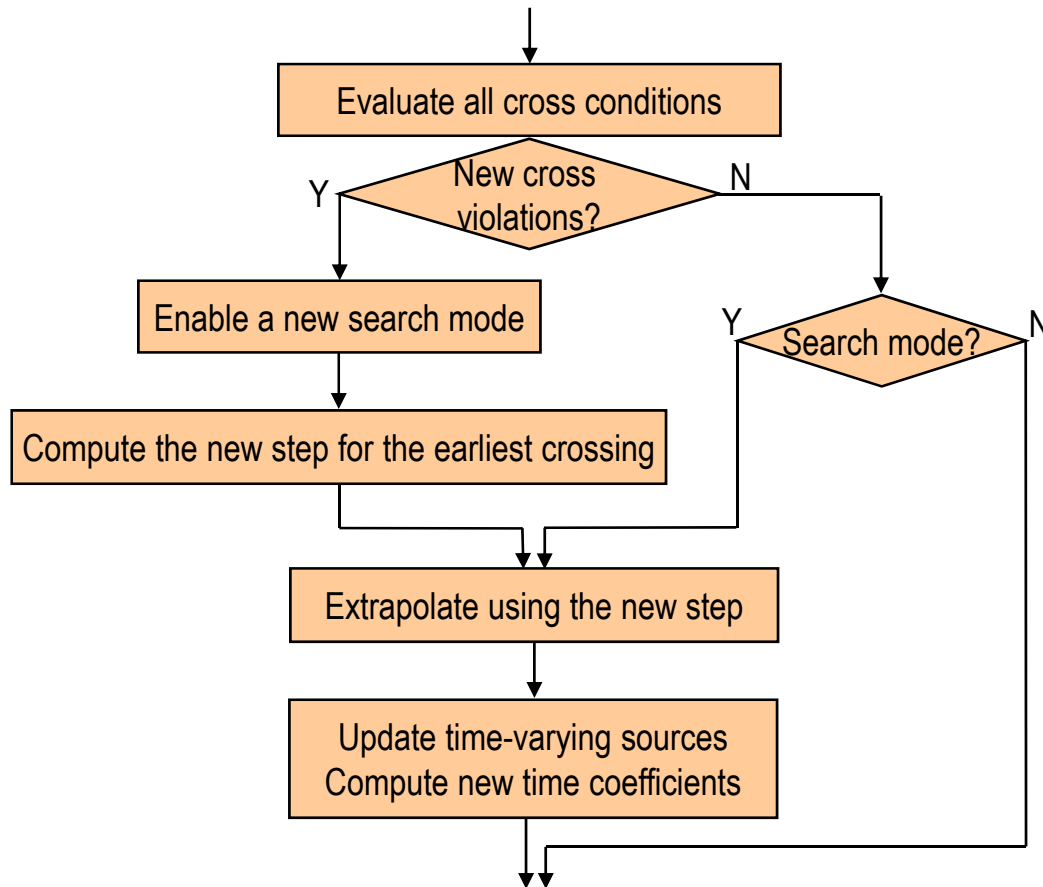
New Transient Flow



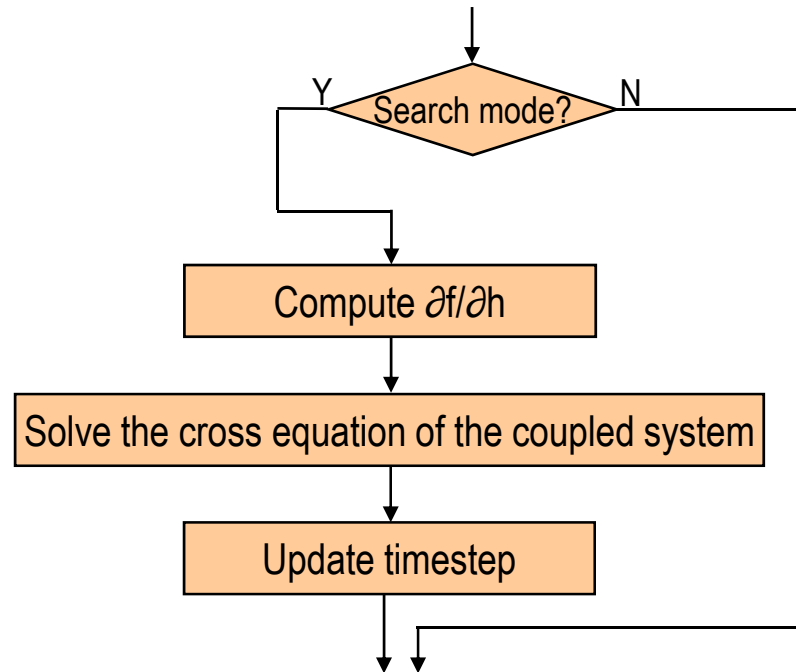
Coupled Newton Flow



Flow for Checking Cross Conditions



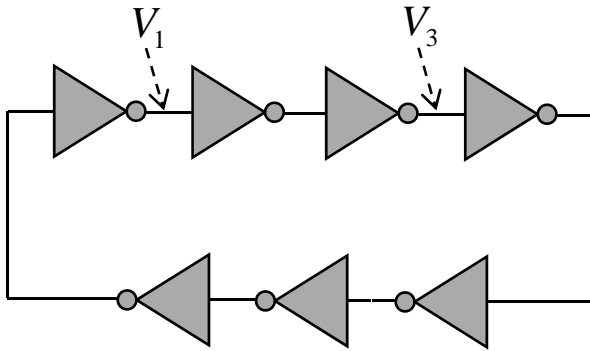
Flow for Solving the Cross Condition



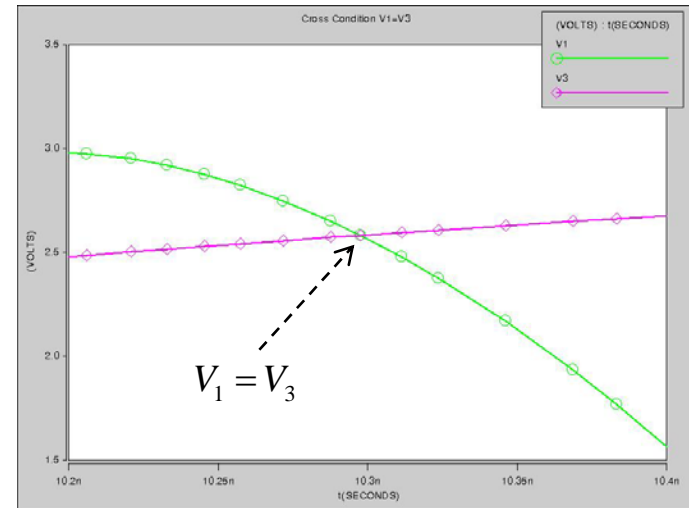
Pros and Cons

- Very low overhead
 - Only a few extra substitutions
- No extra time points needed
 - 2 extra newton iterations vs. 3 extra time point w/ 3 newton iterations at each point = 9 newton iterations
 - Straightforward to implement
 - Clean interface to matrix solvers; works with any type
 - Transparent to the TRAN flow
- Require derivatives of the cross equations
 - Might not be able to handle hidden states (rare)

Simulation Results

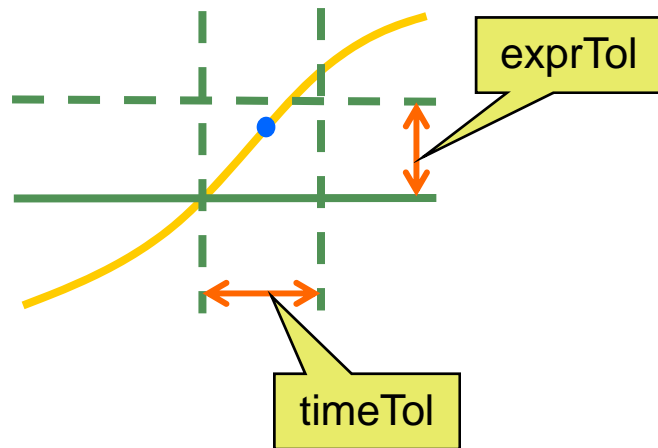


Average # of Newton iterations per accepted timepoint is 2.1



cross condition	average # of Newton iterations at the crossing	average # of extra Newton iterations at the crossing
$V_1 = 2$	3.2	1.1
$V_1 = V_3$	3.5	1.4
$\frac{V_1^2}{1k\Omega} = 4mW$	3.4	1.3

@cross() in Verilog-A/MS



- The cross function requires a timepoint to be placed just after the crossing, within tolerances
- The coupled Newton method needs to be modified slightly to ensure the timepoint placed after the crossing

Summary

- An efficient method based on coupled Newton has been developed to simulate cross events;
- Simulation results show good convergence rates - less than 2 extra Newton iterations needed per crossing;
- This method can be easily integrated into an existing simulator;
- This method can be generalized to solve any monitored events for DAE based systems.