ECE 521 Lecture #13 Nov. 2, 2016

Note Title 11/2/2016

- Stamp for BJT emailed 3(f)
- emailed correct test16.ckt

- Stability of integration methods
- Stiff circuits
  - variable timesteps
  - implicit integration methods

Timestep control

$$h_n \leq \left[\frac{E_n}{\left| C_{k+1}\, x^{(k+1)}(t_n) \right|} \right]^{\frac{1}{k+1}}$$

SPICE uses divided differences for $x^{(k+1)}(t_n)$

Timestep control in SPICE

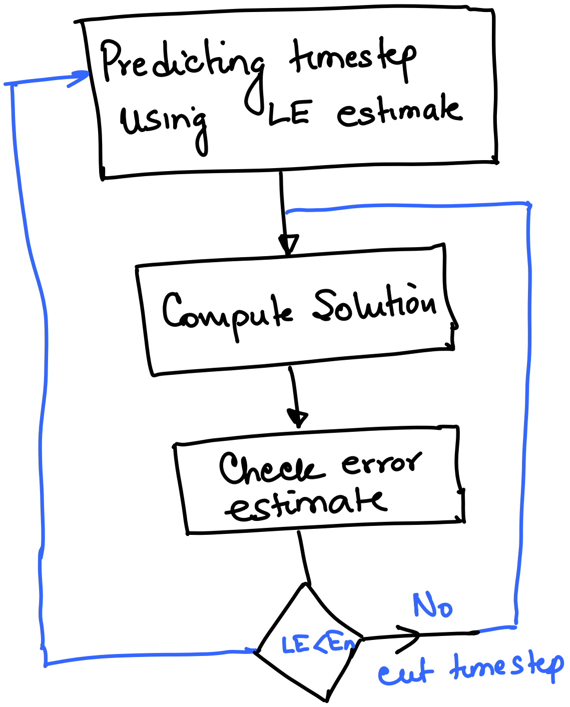
$$\text{.tran}\ \ \underbrace{\text{TSTEP}}_{\substack{\uparrow \\ \text{user specified} \\ \text{time interval}}}\ \ \underbrace{\text{TSTOP}}\ \ \underbrace{\text{TSTART}}_{\substack{\uparrow \\ \text{start point} \\ \text{for outputting} \\ \text{data}}}\ \ \overbrace{\text{TMAX}}^{\text{max } h}$$

$$\text{DELMAX} = \text{MIN}\left(\frac{\text{TSTOP}}{50},\ \text{TMAX},\ \underset{\substack{\uparrow \\ \text{used only when} \\ \text{there are no} \\ \text{energy storage} \\ \text{elements}}}{\text{TSTEP}} \right)$$

$$\text{DELMIN} = 10^{-9} * \text{DELMAX}$$

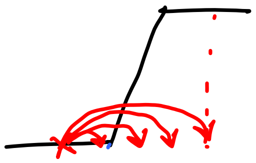
A new timestep is determined from the LE estimate h_n

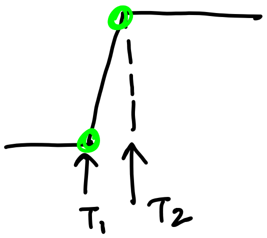
$$h_n = \text{MIN}\left(2 * h_{n-1},\ h_n,\ \text{DELMAX} \right)$$



The timestep is cut by a factor of 8 if the nonlinear equations do not converge in 10 iterations

Sharp input transitions





T_1, T_2 are called breakpoints

- A solution time point is forced at each breakpoint
- A first-order integration method is used after the breakpoint (BE)

Iteration Count timestep Control

Two iteration limits ITL3 (default 4)
ITL4 (default 10)

If Newton's method doesn't converge in ITL4 iterations → cut timestep by a factor of 8

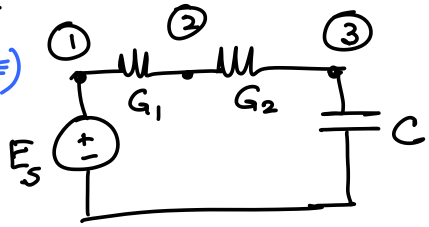
If it converges in less than ITL3 iterations then increase the timestep by a factor of 2.

When can this method be a problem?

Linear circuits

Application to circuits

Upto now $\dot{x} = f(x)$ (ODE)



(A) $G_1(V_1 - V_2) + I_E = 0$

(A) $G_1(V_2 - V_1) + G_2(V_2 - V_3) = 0$

(D) $G_2(V_3 - V_2) + C\frac{dV_3}{dt} = 0$

(A) $V_1 = E_s(t)$

$F(x, \dot{x}, t) = 0$ Differential Algebraic eqns. (DAEs)

At time t_n: $F(x_n, \dot{x}_n, t_n) = 0$

LMS method $\sum_{i=0}^{p} \alpha_i x_{n-i} + h\beta_i \dot{x}_{n-i} = 0$

$$\alpha_0 x_n + h\beta_0 \dot{x}_n + \sum_{i=1}^{p} \alpha_i x_{n-i} + h\beta_i \dot{x}_{n-i} = 0$$

$$\dot{x}_n = -\frac{\alpha_0}{h\beta_0} x_n - \frac{1}{h\beta_0} \underbrace{\sum_{i=1}^{p} \alpha_i x_{n-i} + h\beta_i \dot{x}_{n-i}}_{}$$

$$= \alpha x_n + \beta$$

Our problem to be solved is $F(x_n, \alpha x_n + \beta, t_n) = 0$

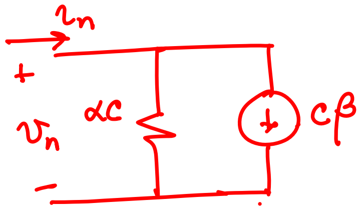
This is a nonlinear equation in unknowns x_n

$\Rightarrow$ Newton's method

- Resistive elements are not affected

- $i = C\frac{dv}{dt}$ linear capacitor

$$i_n = C(\alpha v_n + \beta)$$
$$= \alpha C v_n + C\beta$$



BE: $x_n = x_{n-1} + h\dot{x}_n \rightarrow \dot{x}_n = \frac{x_n - x_{n-1}}{h}$

$$= \alpha x_n + \beta.$$

$$\Rightarrow \alpha = \frac{1}{h}, \quad \beta = \frac{-1}{h}$$

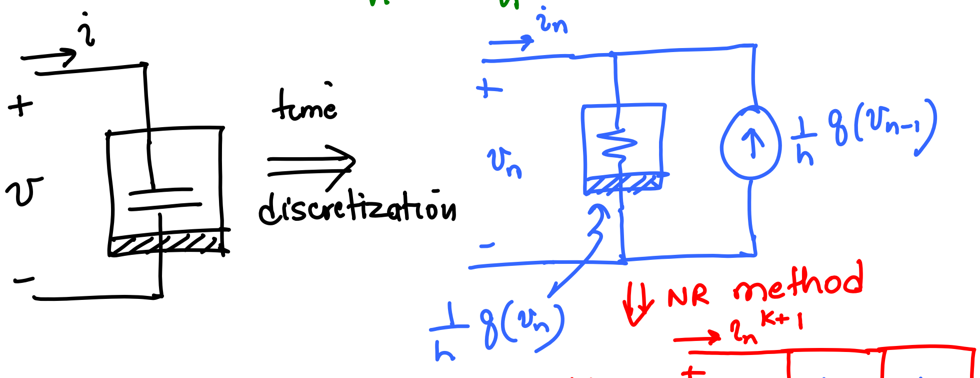
Nonlinear capacitors $\quad q = q(v)$

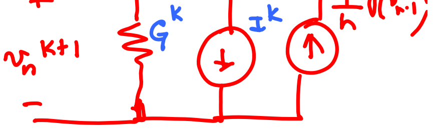
$$i = \frac{dq}{dt} = \overset{=C(v)}{\frac{\partial q}{\partial v}} \frac{dv}{dt}$$ (incremental capacitance)

Capacitance based formulations do not conserve charge so we will focus on a charge based formulation

$$i_n = \alpha q_n + \beta_q = \alpha q(v_n) + \beta_q$$

BE $\quad i_n = \frac{q_n}{h} - \frac{q_{n-1}}{h} = \frac{1}{h}q(v_n) - \frac{1}{h}q(v_{n-1})$



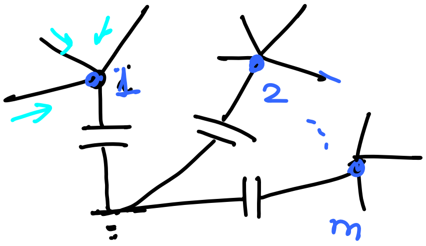


$$i_n^{k+1} = i_n^k + \left.\frac{\partial i_n}{\partial v_n}\right|_{v_n^{k+1}} (v_n^{k+1} - v_n^k)$$

$$G^k = \frac{1}{h}\left.\frac{\partial q(v_n)}{\partial v_n}\right|_{v_n^k}$$

$$I^k = \frac{q(v_n^k)}{h} - G_k v_n^k$$

Charge conservation



$$\frac{dq_i}{dt} = f_i(v_i)$$

$$C(v_i)\frac{dv_i}{dt} = f_i(v_i)$$

where $C(v_i) = \left.\frac{\partial q}{\partial v}\right|_{v_i}$

Charge conservation ⇒ $\sum_{i=1}^{m+1} q_i(v) = K$

$$\sum_{i=1}^{m+1} f_i(v) = 0$$

Using FE as the integration method

$$\frac{dq_i}{dt} = f_i(v)$$

$$\frac{q_n - q_{n-1}}{h} = f_i(v_{n-1})$$

$$\sum_{i=1}^{m+1} q_{i,n} - \sum_{i=1}^{m+1} q_{i,n-1} = h\cancelto{0}{\sum_{i=1}^{m+1} f_i(v_{n-1})}$$

$$\sum_{i=1}^{m+1} q_{i,n} = \sum_{i=1}^{m+1} q_{i,n-1} = K$$

i.e. charge is conserved

Using the capacitance formulation

$$C_i(v_{n-1})\,\frac{\underset{i}{v_n} - \underset{i}{v_{n-1}}}{h} = f_i(v_{n-1})$$

$$\sum_{i=1}^{m+1} q(v_n) = \sum_{i=1}^{m+1} q(v_{n-1}) + \cancelto{0}{\sum_{i=1}^{m+1} \frac{\partial q}{\partial v}\Big|_{v_{n-1}}} (v_n - v_{n-1})$$

$$+ \sum_{i=1}^{m+1} \frac{1}{2} \frac{\partial^2 q}{\partial v^2}\Big|_{v=\hat{v}} (v_n - v_{n-1})^2$$

$$\frac{\partial q}{\partial v}\Big|_{v_{n-1}} (v_n - v_{n-1}) = C(v_{n-1})(v_n - v_{n-1})$$

$$= h f_i(v_{n-1})$$

This shows that charge is not conserved

<u>Observations on BDF</u>

$$\dot{x}_n = -\frac{1}{h_n} \sum_{i=0}^{k} \alpha_i x_{n-i} \qquad \text{Kth order BDF}$$

Kth order predictor: $x_n^0 = \sum_{i=1}^{k+1} \gamma_i x_{n-i} = \underset{\overset{\uparrow}{\text{predicted}}}{x_n^p}$

It can be shown that

$$LE_n = \frac{h_n}{t_n - t_{n-k-1}} \left(\underset{\overset{\uparrow}{\text{Corrected}}}{x_n^c} - x_n^p \right)$$

<u>So far</u>: DC solution of linear, nonlinear circuits
transient analysis " "

Inside a Circuit Simulator – SPICE3

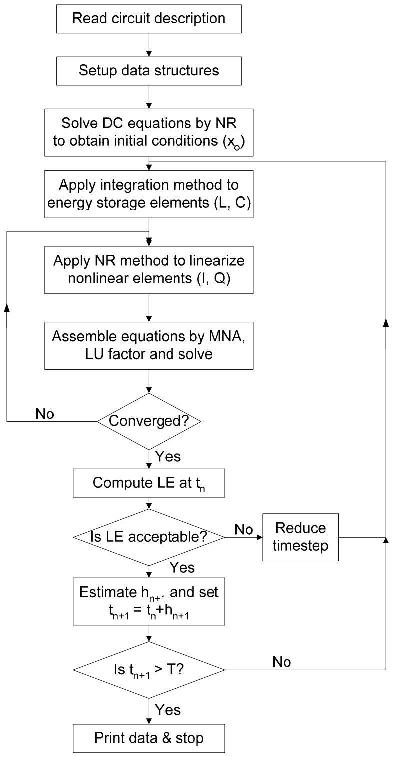
- **SPICE2 is outdated**
  - 20K lines of FORTRAN
  - 10 years of changes (1985 – Version 2G6)
  - Basic algorithms, devices, data structures distributed throughout code
  - Difficult to add new models, analyses
  - Difficult to maintain

SPICE3 (1985) – Modular/Extensible Version

- **Faster**
- **More robust**
  - Model problems/discontinuities removed
- **Flexible framework for circuit simulation**
  - New models and analyses easily added
- **New and improved algorithms**
  - Gmin and source stepping
  - Predictor-corrector integration (BDF)
- **New device models**
  - GaAs MESFET, BSIM1, BSIM3
  - Voltage/current controlled switches
  - Arbitrary controlled sources
  - Uniform distributed R-C lines
- **Clean simulator interfaces**
  - Clearly defined functions, independent of frontend
  - Table driven

Flowchart for a circuit simulator

Transient analysis



Read circuit description

Setup data structures

Solve DC equations by NR to obtain initial conditions (x_o)

Apply integration method to energy storage elements (L, C)

Apply NR method to linearize nonlinear elements (I, Q)

Assemble equations by MNA, LU factor and solve

Converged?

No

Yes

Compute LE at t_n

Is LE acceptable?

No

Reduce timestep

Yes

Estimate h_{n+1} and set $t_{n+1} = t_n+h_{n+1}$

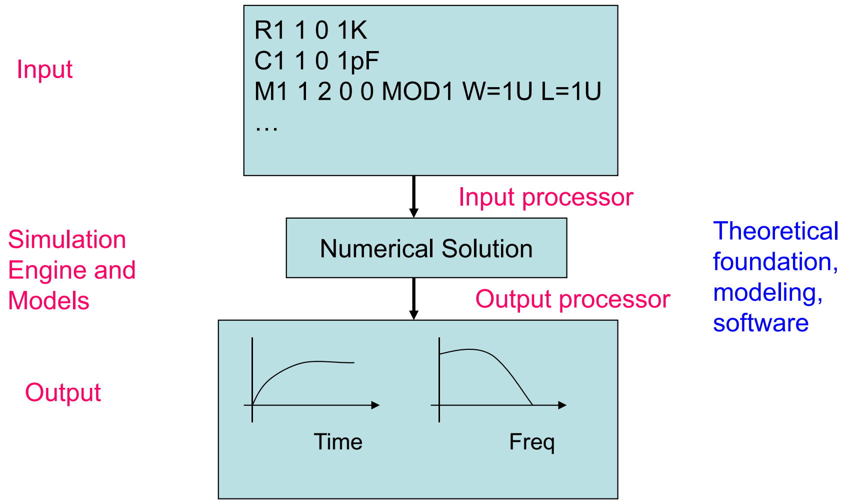
Is t_{n+1} > T?

No

Yes

Print data & stop

Components of SPICE



Input

R1 1 0 1K
C1 1 0 1pF
M1 1 2 0 0 MOD1 W=1U L=1U
…

Input processor

Simulation Engine and Models

Numerical Solution

Output processor

Theoretical foundation, modeling, software

Output

Required Components

- Input processor
- Data structures
- Theoretical foundations
- Simulation engine
- Output processor

Theoretical foundations

- Equation formulation
  - MNA
- Linear equation solution
  - LU factorization, forward/back solve
- Solution of nonlinear equations, convergence
  - Newton's method
- Numerical solution of differential equations
  - Integration methods
    - Accuracy
    - Stability
    - Variable timesteps

Simulation Engine

- Numerical algorithms
- Models for devices
- Control loops for analyses
- Analyses
  - DC, OP
  - TRAN
  - AC
  - FOUR
  - NOISE
  - …

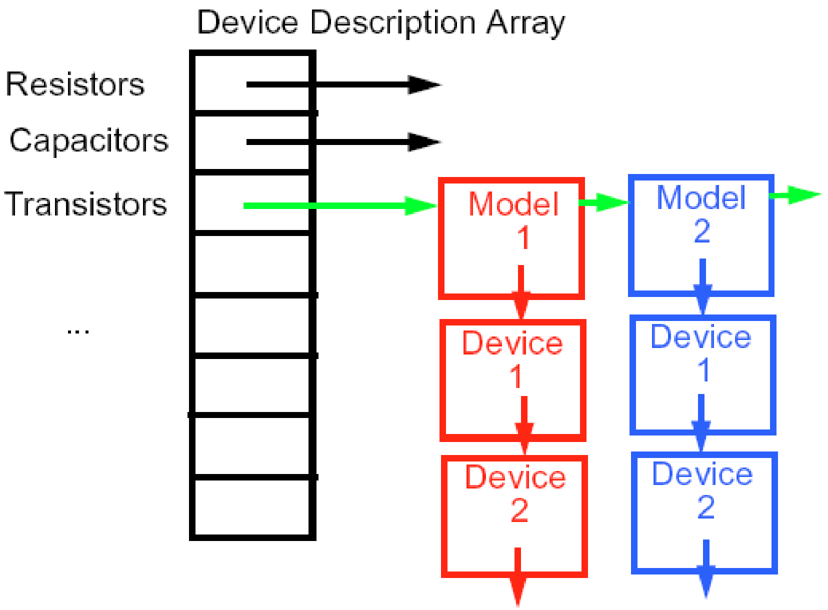
Output Processor

- Save solution
- Plotting and printing of solution
- Provide post processing capabilities
  - Compute i*v, v1-v2-v3, …
  - Functions of output variables
  - Fourier analysis etc.

SPICE3 – Building Blocks

- Newton's method for nonlinear equation solution
- Sparse1.3 for linear equation solution
- Integration methods: TR, Gear (BDF)
- Variable timestep/order control
- Postprocessor - Nutmeg

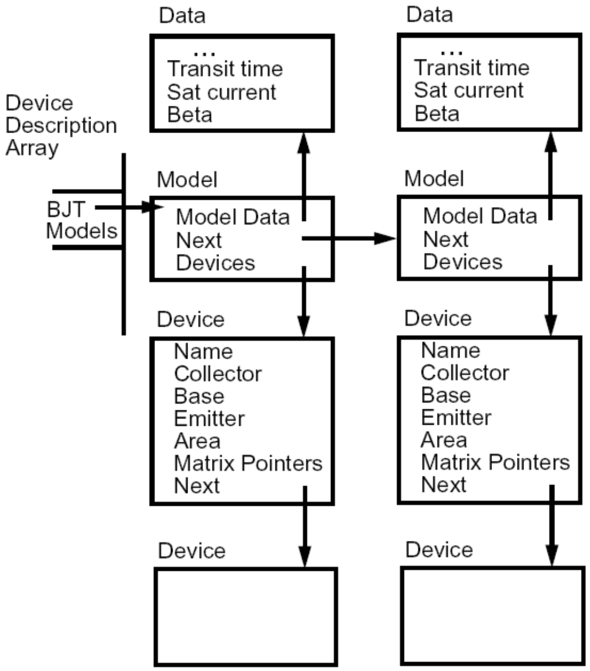
Circuit Data Structures – SPICE3



Advantages

- Store generic device information in model instead of device
- Preprocess model and then devices associated with model
- Can easily skip devices

SPICE3 – Device Data Structures



SPICE3 – Resdefs.h (Device)

```
/* information used to describe a single instance */
typedef struct sRESinstance {
    struct sRESmodel *RESmodPtr;    /* backpointer to model */
    struct sRESinstance *RESnextInstance;   /* pointer to next instance of
                                             * current model*/

    IFuid RESname;  /* pointer to character string naming this instance */
    int RESstate; /* not used */
    int RESposNode; /* number of positive node of resistor */
    int RESnegNode; /* number of negative node of resistor */

    double REStemp;     /* temperature at which this resistor operates */
    double RESconduct;  /* conductance at current analysis temperature */
    double RESresist;   /* resistance at temperature Tnom */
    double RESwidth;    /* width of the resistor */
    double RESlength;   /* length of the resistor */
    double *RESposPosptr;    /* pointer to sparse matrix diagonal at
                              * (positive,positive) */
    double *RESnegNegptr;    /* pointer to sparse matrix diagonal at
                              * (negative,negative) */
    double *RESposNegptr;    /* pointer to sparse matrix offdiagonal at
                              * (positive,negative) */
    double *RESnegPosptr;    /* pointer to sparse matrix offdiagonal at
                              * (negative,positive) */
    unsigned RESresGiven : 1;   /* flag to indicate resistance was specified */
    unsigned RESwidthGiven  : 1;    /* flag to indicate width given */
    unsigned RESlengthGiven : 1;    /* flag to indicate length given */
    unsigned REStempGiven   : 1;    /* indicates temperature specified */
 int    RESsenParmNo;         /* parameter # for sensitivity use;
                set equal to  0 if not a design parameter*/
} RESinstance ;
```

SPICE3 – Resdefs.h (Model)

```
/* per model data */

typedef struct sRESmodel {       /* model structure for a resistor */
    int RESmodType; /* type index of this device type */
    struct sRESmodel *RESnextModel; /* pointer to next possible model in
                                     * linked list */
    RESinstance * RESinstances; /* pointer to list of instances that have this
                                 * model */
    IFuid RESmodName;       /* pointer to character string naming this model */

    double REStnom;         /* temperature at which resistance measured */
    double REStempCoeff1;   /* first temperature coefficient of resistors */
    double REStempCoeff2;   /* second temperature coefficient of resistors */
    double RESsheetRes;     /* sheet resistance of devices in ohms/square */
    double RESdefWidth;     /* default width of a resistor */
    double RESnarrow;       /* amount by which device is narrower than drawn */
    unsigned REStnomGiven: 1;   /* flag to indicate nominal temp. was given */
    unsigned REStc1Given : 1;   /* flag to indicate tc1 was specified */
    unsigned REStc2Given : 1;   /* flag to indicate tc2 was specified */
    unsigned RESsheetResGiven   :1; /* flag to indicate sheet resistance given*/
    unsigned RESdefWidthGiven   :1; /* flag to indicate default width given */
    unsigned RESnarrowGiven     :1; /* flag to indicate narrow effect given */
} RESmodel;
```

SPICE3 – Ressetup.c

```
#include "spice.h"
#include <stdio.h>
#include "util.h"
#include "smpdefs.h"
#include "resdefs.h"
#include "sperror.h"
#include "suffix.h"

int
RESsetup(matrix,inModel,ckt,state)
    register SMPmatrix *matrix;
    GENmodel *inModel;
    CKTcircuit*ckt;
    int *state;
        /* load the resistor structure with those pointers needed later
         * for fast matrix loading
         */
{
    register RESmodel *model = (RESmodel *)inModel;
    register RESinstance *here;
 /*  loop through all the resistor models */
    for( ; model != NULL; model = model->RESnextModel ) {

        /* loop through all the instances of the model */
        for (here = model->RESinstances; here != NULL ;
                here=here->RESnextInstance) {

/* macro to make elements with built in test for out of memory */
#define TSTALLOC(ptr,first,second) \
if((here->ptr = SMPmakeElt(matrix,here->first,here->second))==(double *)NULL){\
    return(E_NOMEM);}
            TSTALLOC(RESposPosptr, RESposNode, RESposNode);
            TSTALLOC(RESnegNegptr, RESnegNode, RESnegNode);
            TSTALLOC(RESposNegptr, RESposNode, RESnegNode);
            TSTALLOC(RESnegPosptr, RESnegNode, RESposNode);
        }
    }
    return(OK);
}
```

SPICE3 – Resload.c

```
#include "spice.h"
#include <stdio.h>
#include "cktdefs.h"
#include "resdefs.h"
#include "sperror.h"
#include "suffix.h"

/*ARGSUSED*/
int
RESload(inModel,ckt)
    GENmodel *inModel;
    CKTcircuit *ckt;
        /* actually load the current resistance value into the
         * sparse matrix previously provided
         */
{
    register RESmodel *model = (RESmodel *)inModel;
    register RESinstance *here;

    /*  loop through all the resistor models */
    for( ; model != NULL; model = model->RESnextModel ) {
 /* loop through all the instances of the model */
        for (here = model->RESinstances; here != NULL ;
                here=here->RESnextInstance) {

            *(here->RESposPosptr) += here->RESconduct;
            *(here->RESnegNegptr) += here->RESconduct;
            *(here->RESposNegptr) -= here->RESconduct;
            *(here->RESnegPosptr) -= here->RESconduct;
        }
    }
    return(OK);
}
```

SPICE3 – Vsrcdefs.h (Device)

```
typedef struct sVSRCinstance {
    struct sVSRCmodel *VSRCmodPtr;  /* backpointer to model */
    struct sVSRCinstance *VSRCnextInstance;  /* pointer to next instance of
                                              *current model */
    IFuid VSRCname; /* pointer to character string naming this instance */
    int VSRCstate;      /* not used */
    int VSRCposNode;    /* number of positive node of resistor */
    int VSRCnegNode;    /* number of negative node of resistor */
    int VSRCbranch; /* equation number of branch equation added for source */
    int VSRCfunctionType;   /* code number of function type for source */
    int VSRCfunctionOrder;  /* order of the function for the source */
    double *VSRCcoeffs; /* pointer to array of coefficients */

    double VSRCdcValue; /* DC and TRANSIENT value of source */

    double VSRCacReal; /* AC real component */
    double VSRCacImag; /* AC imaginary component */

    ...
    double *VSRCposIbrptr;  /* pointer to sparse matrix element at
                             * (positive node, branch equation) */
    double *VSRCnegIbrptr;  /* pointer to sparse matrix element at
                             * (negative node, branch equation) */
    double *VSRCibrPosptr;  /* pointer to sparse matrix element at
                             * (branch equation, positive node) */
    double *VSRCibrNegptr;  /* pointer to sparse matrix element at
                             * (branch equation, negative node) */
    double *VSRCibrIbrptr;  /* pointer to sparse matrix element at
                             * (branch equation, branch equation) */
    unsigned VSRCdcGiven     :1 ;   /* flag to indicate dc value given */
    ...
} VSRCinstance ;
```

SPICE3 – Vsrcdefs.h (Model)

```
/* per model data */

typedef struct sVSRCmodel {
    int VSRCmodType;    /* type index of this device type */
    struct sVSRCmodel *VSRCnextModel;    /* pointer to next possible model
                                          *in linked list */
    VSRCinstance * VSRCinstances;    /* pointer to list of instances
                                      * that have this model */
    IFuid VSRCmodName;       /* pointer to character string naming this
model */
} VSRCmodel;
```

SPICE3 – Vsrcsetup.c

```
int
VSRCsetup(matrix,inModel,ckt,state)
    register SMPmatrix *matrix;
    GENmodel *inModel;
    register CKTcircuit *ckt;
    int *state;
        /* load the voltage source structure with those pointers needed later
         * for fast matrix loading
         */
{
    register VSRCmodel *model = (VSRCmodel *)inModel;
    register VSRCinstance *here;
    CKTnode *tmp;
    int error;
/*  loop through all the voltage source models */
    for( ; model != NULL; model = model->VSRCnextModel ) {

        /* loop through all the instances of the model */
        for (here = model->VSRCinstances; here != NULL ;
                here=here->VSRCnextInstance) {

            if(here->VSRCbranch == 0) {
                error = CKTmkCur(ckt,&tmp,here->VSRCname,"branch");
                if(error) return(error);
                here->VSRCbranch = tmp->number;
            }
/* macro to make elements with built in test for out of memory */
#define TSTALLOC(ptr,first,second) \
if((here->ptr = SMPmakeElt(matrix,here->first,here->second))==(double *)NULL){\
    return(E_NOMEM);\
}

            TSTALLOC(VSRCposIbrptr, VSRCposNode, VSRCbranch)
            TSTALLOC(VSRCnegIbrptr, VSRCnegNode, VSRCbranch)
            TSTALLOC(VSRCibrNegptr, VSRCbranch, VSRCnegNode)
            TSTALLOC(VSRCibrPosptr, VSRCbranch, VSRCposNode)
        }
    }
    return(OK);
```

SPICE3 – Vsrcload.c

```
int
VSRCload(inModel,ckt)
    GENmodel *inModel;
    register CKTcircuit *ckt;
{
    register VSRCmodel *model = (VSRCmodel *)inModel;
    register VSRCinstance *here;
    double time;
    for( ; model != NULL; model = model->VSRCnextModel ) {
        for (here = model->VSRCinstances; here != NULL ;
                here=here->VSRCnextInstance) {
            *(here->VSRCposIbrptr) += 1.0 ;
            *(here->VSRCnegIbrptr) -= 1.0 ;
            *(here->VSRCibrPosptr) += 1.0 ;
            *(here->VSRCibrNegptr) -= 1.0 ;
            if( (ckt->CKTmode & (MODEDCOP | MODEDCTRANCURVE)) &&
                    here->VSRCdcGiven ) {
                /* grab dc value */
                *(ckt->CKTrhs + (here->VSRCbranch)) += ckt->CKTsrcFact *
                        here->VSRCdcValue;
            } else {
 if(ckt->CKTmode & (MODEDC)) {
                    time = 0;
                } else {
                    time = ckt->CKTtime;
                }
                /* use the transient functions */
                switch(here->VSRCfunctionType) {
                default: { /* no function specified:   use the DC value */
                    *(ckt->CKTrhs + (here->VSRCbranch)) += here->VSRCdcValue;
                    break;
                }
                case PULSE:
                case SINE:
                case PWL:
            }
loadDone: ;
        }
    }
    return(OK);
}
```

Device Description Array

An array of structures that contains all information
the simulator core needs to know about devices

Name
∗pModel /∗ pointer to model / device list ∗/
(∗pCreateModel)( )
(∗pCreateInstance)( )
(∗pSetUp)( )
(∗pSetTemperature)( )
(∗pDC_Load)( )
(∗pAC_Load)( )
(∗pTransientLoad)( )
(∗pOpPoint)( )

Device
Function
Pointers

Nodes
DeviceParamCount
DeviceParams
ModelParamCount
ModelParams

Parser
Information

Syntax

help information

Parameter Data Structure

An array that describes each device or model parameter

BJT_params[ ] = {
{ 0, “bf”, “beta forward”};
{ 1, “is”, “saturation current”};
{ 2, “tf”, “forward transit time”};
}

internal code | external name | parameter description for help

SPICE - Input Phase

- **Information read in**
  - Devices in circuit (R, C, M, …)
  - Models for devices (NMOS, PMOS, …)
  - Analysis requests (.DC, .TRAN, …)
  - Analysis parameters
    - **TSTOP, TSTEP, TMAX, …**
    - **RELTOL, ABSTOL**
  - Output requests (.PRINT, .PLOT)

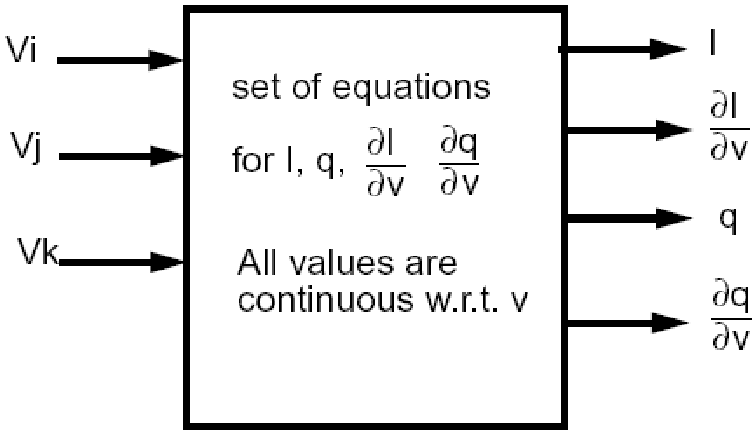
SPICE – Setup Phase

- **Preprocess all models**
  - Set all default values
  - Check validity of user specified values
  - Calculate useful quantities (e.g., RD -> GD)
- **Preprocess all devices**
  - Allocate matrix entries, establish pointers
  - Calculate useful quantities (e.g., k’W/L)
- **Circuit topology checks**
  - DC path to ground from all nodes
  - All nodes have at least two connections
  - …

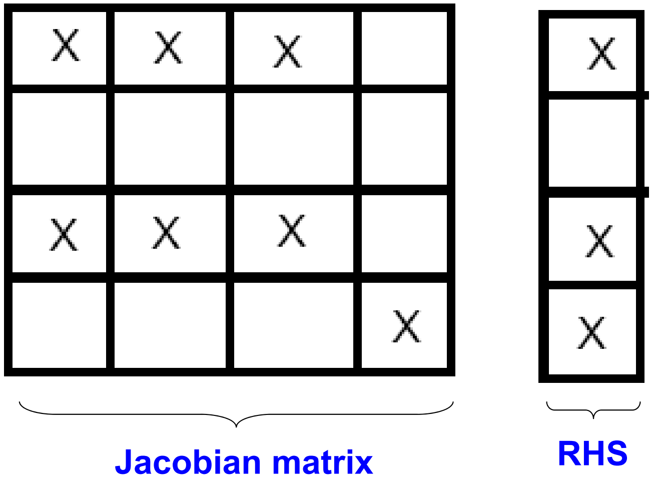
Post Setup

- **Circuit is topologically correct**
- **Model and device parameters are reasonable**
- **Precalculated model/device quantities stored**
- **Matrix allocated and direct pointers established**
- **Circuit data structures allocated**

Device Evaluation Functions


Vi
Vj
Vk
set of equations
for I, q, $\frac{\partial I}{\partial v}$ $\frac{\partial q}{\partial v}$
All values are continuous w.r.t. v
I
$\frac{\partial I}{\partial v}$
q
$\frac{\partial q}{\partial v}$

Matrix Load (Stamping)



$$\underbrace{\begin{array}{|c|c|c|c|}\hline X & X & X & \\ \hline & & & \\ \hline X & X & X & \\ \hline & & & X \\ \hline\end{array}}_{\text{Jacobian matrix}} \quad \underbrace{\begin{array}{|c|}\hline X \\ \hline \\ \hline X \\ \hline X \\ \hline\end{array}}_{\text{RHS}}$$