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

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

Timestep control

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

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

---

**Timeskip control in SPICE**

\[ ^{\text{tran}} \] TSTEP TSTOP TSTART TMAX

\[ \text{User specified time interval} \]

\[ \text{Start point for outputting data} \]

\[ \text{DELMAX} = \text{MIN} \left( \frac{\text{TSTOP}}{50}, \text{TMAX}, \text{TSTEP} \right) \]

\[ \text{DELMIN} = 10^{-9} \times \text{DELMAX} \]

\[ \text{Used only when there are no energy storage elements} \]

A new timestep is determined from the LE estimate \( h \)

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

<table>
<thead>
<tr>
<th>Two iteration limits</th>
<th>ITL3 (default 4)</th>
<th>ITL4 (default 10)</th>
</tr>
</thead>
</table>

If Newton's method doesn't converge in ITL4 iterations $\rightarrow$ 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) \( V_1 \)

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

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

LMS method

\[
\sum_{i=0}^{p} \alpha_i x_{n-i} + h \beta_i \dot{x}_{n-i} = 0
\]

\[
\dot{x}_n = \frac{-d_0 x_n - \frac{1}{h \beta_0} \sum_{i=1}^{p} \alpha_i x_{n-i} + h \beta_i \dot{x}_{n-i}}{h \beta_0}
\]

\[
= \alpha x_n + \beta
\]

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
  \[ l_n = C \left( \frac{d}{dt} v_n + \beta \right) = -\frac{1}{C} \frac{dv_n}{dt} + C \beta \]
  \[ BE: \quad x_n = x_{n-1} + h \dot{x}_n \quad \Rightarrow \quad \dot{x}_n = \frac{x_n - x_{n-1}}{h} = \dot{x}_n + \beta. \]

  \( \Rightarrow \quad \dot{x} = \frac{1}{h}, \quad \beta = -\frac{1}{h} \)

**Nonlinear capacitors**

\[ g = g(v) \]

\[ i = \frac{dq}{dt} = \frac{\partial g}{\partial v} \frac{dv}{dt} \]

\[ (\text{Incremental Capacitance}) \]

---

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

\[ l_n = \omega q_n + \beta q = \omega g(v_n) + \beta g \]

**BE**

\[ \dot{i}_n = \frac{q_n}{h} - \frac{q_{n-1}}{h} = \frac{1}{h} g(v_n) - \frac{1}{h} g(v_{n-1}) \]

\[ \text{Time discretization} \]

\[ \dot{i}_n = \frac{1}{h} g(v_n) \]

\[ \text{NR method} \]

\[ l_{n+1} = l_n + \frac{\partial l}{\partial l_n} \bigg|_{v_{n+1}^k - v_n} \]

\[ v_{n+1}^k = v_n + \frac{1}{h} g(v_n) \]

\[ \text{NR method} \]

\[ l_{n+1} = l_n + \frac{\partial l}{\partial l_n} \bigg|_{v_{n+1}^k - v_n} \]

\[ v_{n+1}^k = v_n + \frac{1}{h} g(v_n) \]
\[ G^k = \frac{1}{\hbar} \left. \frac{\partial g(v_n)}{\partial \nu_n} \right|_{\nu_n^k} \]

\[ I^k = g(v_n^k) - G^k \nu_n^k \]

Charge conservation:

\[ \frac{dq_i}{dt} = f_i(v_i) \]

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

where \[ C(v_i) = \left. \frac{\partial g}{\partial v} \right|_{v_i} \]

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

\[ \frac{dq_i}{dt} = f_i(v) \]

\[ \sum_{i=1}^{m+1} q_i - q_{n-1} = f_i(v_{n-1}) \]

\[ \sum_{i=1}^{m+1} q_i - q_{n-1} = \sum_{i=1}^{m+1} f_i(v_{n-1}) \]

\[ \sum_{n=1}^{m+1} q_{i,n} - q_{i,n-1} = \sum_{n=1}^{m+1} f_i(v_{n-1}) \]

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

i.e. charge is conserved

Using the capacitance formulation

\[ C_i(v_{n-1}) \frac{v_n - v_{n-1}}{\hbar} = f_i(v_{n-1}) \]
\[
\sum_{i=1}^{m+1} q_i(x_n) = \sum_{i=1}^{m+1} q_i(x_{n-i}) + \sum_{i=1}^{m+1} \frac{\partial y}{\partial v} \bigg|_{v_{n-i}} (v_n - v_{n-1})
\]

\[
+ \sum_{i=1}^{m+1} \frac{2}{\theta v^2} \bigg|_{v_{n-i}} (v_n - v_{n-1})^2
\]

\[
\frac{\partial q}{\partial v} \bigg|_{v_{n-1}} (v_n - v_{n-1}) = C(v_{n-i})(v_n - v_{n-1})
\]

\[
= h \cdot f_i(v_{n-i})
\]

This shows that charge is not conserved.

**Observations on BDF**

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

**K\text{th order predictor}**: \[
x_n^p = \sum_{i=1}^{k+1} v_i x_{n-i} = \uparrow \text{predicted}
\]

It can be shown that

\[
LE_n = \frac{h_n}{t_n - t_{n-k-1}} (x_n^c - x_n^p)
\]

So far: 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

Components of SPICE

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

Input processor

Input

Simulation Engine and Models

Output processor

Numerical Solution

Output

Theoretical foundation, modeling, software

Transient analysis

Read circuit description

Setup data structures

Solve DC equations by NR to obtain initial conditions (v0)

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

Apply NR method to linearize nonlinear elements (i, G)

Assemble equations by MNA, LU factor and solve

Converged?

Yes

No

Compute LE at L

L.E acceptable?

Yes

No

Estimate L.L and set t.L = L.L

L > L?

Yes

No

Print data & stop

Time

Freq
# 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
 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 list */
 IFuid RESmodName; /* pointer to character string naming this model */
 double RESdtnom; /* temperature at which resistance measured */
 double RESdttempCoeff1; /* first temperature coefficient of resistors */
 double RESdttempCoeff2; /* second temperature coefficient of resistors */
 double RESseetRes; /* sheet resistance of devices in ohms/square */
 double RESdtWidth; /* default width of a resistor */
 double RESdtlength; /* amount by which device is narrower than drawn */
 unsigned RESdtStcliven; /* flag to indicate nominal temp. was given */
 unsigned RESdtStcligiven; /* flag to indicate tc1 was specified */
 unsigned RESdtStclgivendefault; /* flag to indicate default tc1 was given */
 unsigned RESdtStchlength given; /* flag to indicate sheet resistance given */
 unsigned RESdtStchwidthgiven; /* flag to indicate default width given */
 unsigned RESdtStchnarrowgiven; /* flag to indicate narrow effect given */
 } RESmodel;

 int RESsetup(matrix,inModel,ckt,state) 
{ 
 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); 
} 

 int RESload(inModel,ckt) 
{ 
 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);
} 
*(here->RESposPosptr) += here->RESconduct;*(here->RESnegNegptr) += here->RESconduct;*(here->RESposNegptr) += here->RESconduct;*(here->RESnegPosptr) += here->RESconduct; 
} 
return(OK); 
}
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 */
...}

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;

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 for matrix loading*/
{
    register VSRCmodel *model = (VSRCmodel *)inModel;register VSRCinstance *here;CKTnode *tmp;
/*  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);
} else {
    time = ((double *)here->ptr)->time;
    if(time < 0) {
        if((here->VSRCbranch >= 0)) {  
            error = VCUpkCur(ckt->ckt,hp,here->VSRCname,"branch");
            if(error) return(error);
        }
    }  
    /* use the transient functions */
    switch(here->VSRCfunctionType) {
    default: {
        /* no function specified: use the DC value */
        *(ckt->CKTrhs + (here->VSRCbranch)) += ckt->CKTsrcFact * here->VSRCdcValue;
        break;
    } case PULSE: case SINE: case PWL: case (...) loadDone: ;
    }
    return(OK);
  }
}
Device Description Array

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

- `Name`
- `Model`
  - `CreateModel()`
  - `CreateInstance()`
  - `SetTemperature()`
  - `DC_Load()`
  - `AC_Load()`
  - `TransientLoad()`
  - `PopPoint()`
- `Device`
- `Function`
- `Points`

Parser

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
  - …

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)

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 → set of equations for I, q, \( \frac{\partial l}{\partial v} \), \( \frac{\partial q}{\partial v} \) → I

Vj → \( \frac{\partial l}{\partial v} \) → \( \frac{\partial q}{\partial v} \) → q

Vk → \( \frac{\partial q}{\partial v} \) → RHS

Matrix Load (Stamping)

Jacobian matrix

RHS