ECE61016 Power Electronic Converters and Systems

Lecture Set 1: Time-Domain Simulation

S.D. Sudhoff

Fall 2016

Some Definitions

- Model – A description of a system used to predict it's behavior

- Simulation – The process of predicting a system's behavior from the model

Time-Domain Simulation

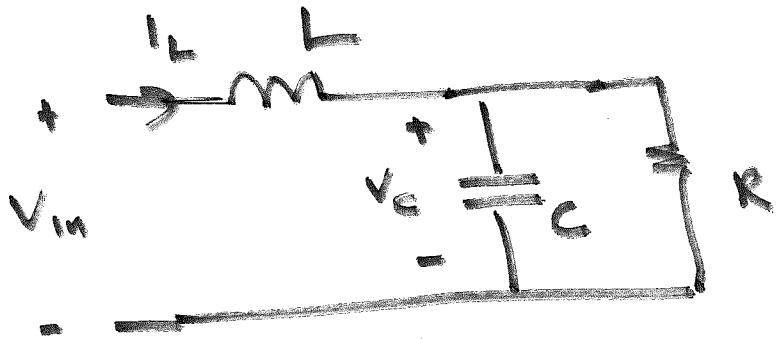
- Resistor-Companion (Circuit) Simulation
  - SPICE, PSPICE, SABRE,PSCAD

- State-Variable (System) Simulation
  - Matlab, Simulink, ACSL, EASY5, Dymola, PLECS

Resistor-Companion Simulation [1]

$V_{x,n}$ = voltage on 'x' at time t_n

$I_{x,n}$ = current on 'x' at time t_n



$$L\frac{dI_L}{dt} = V_{in} - V_c$$

$$\frac{dI_L}{dt} = \frac{1}{L}(V_{in} - V_c)$$

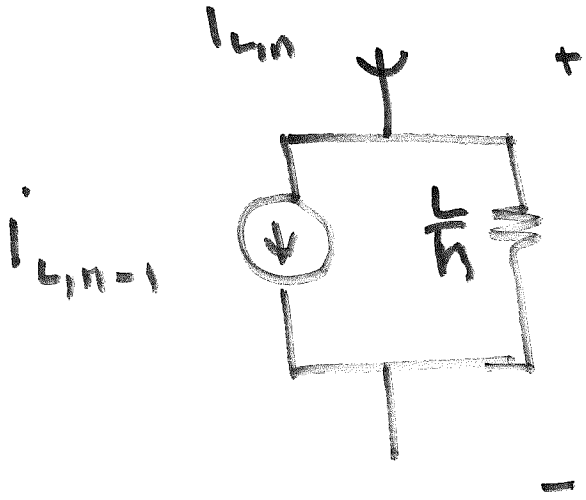
$$I_{L,n} = I_{L,n-1} + \left.\frac{dI_L}{dt}\right|_n \overbrace{(t_n - t_{n-1})}^{h}$$

Backwards Euler

[1] L.O. Chua, P.M. Lin, *Computer Aided Analysis of Electric Circuits: Algorithms & Computation Techniques,* Prentice-Hall, 1975.

Resistor-Companion Simulation [1]

$$I_{L,n} = I_{L,n-1} + \frac{1}{L}\underbrace{(V_{m,n} - V_{c,n})}_{V_{L,n}} h$$



$$V_{L,n} = V_{m,n} - V_{c,n}$$

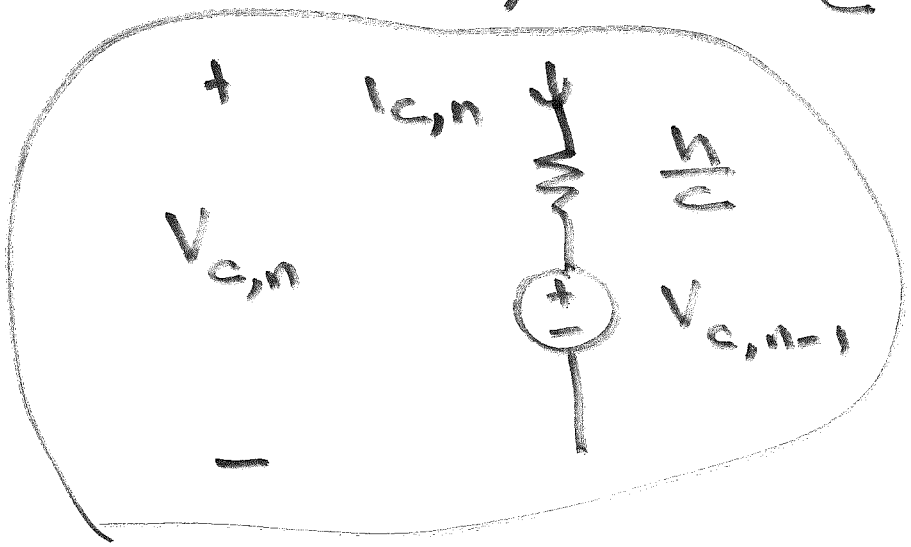
Resistor-Companion Simulation [1]

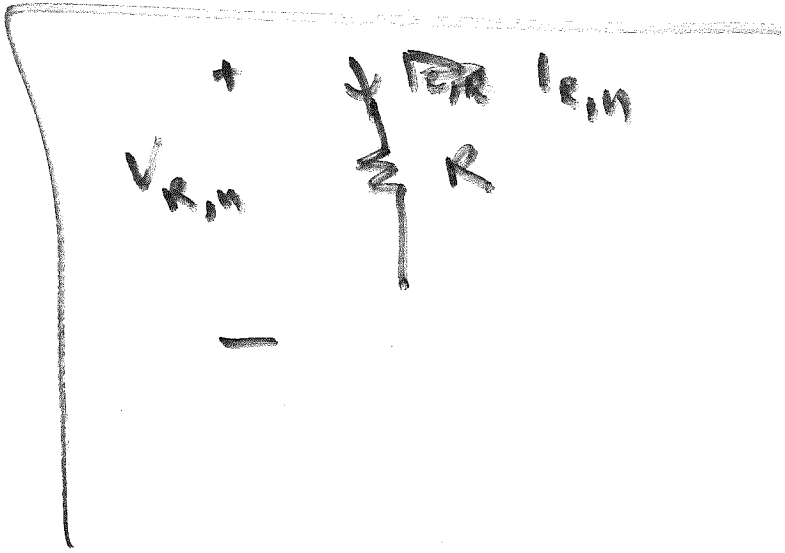
$$C\frac{dV_c}{dt} = I_L - \frac{V_c}{R}$$

$$V_{c,n} = V_{c,n-1} + \frac{dV_c}{dt}\Big|_n (t_n - t_{n-1}) \qquad (t_n - t_{n-1} = h)$$

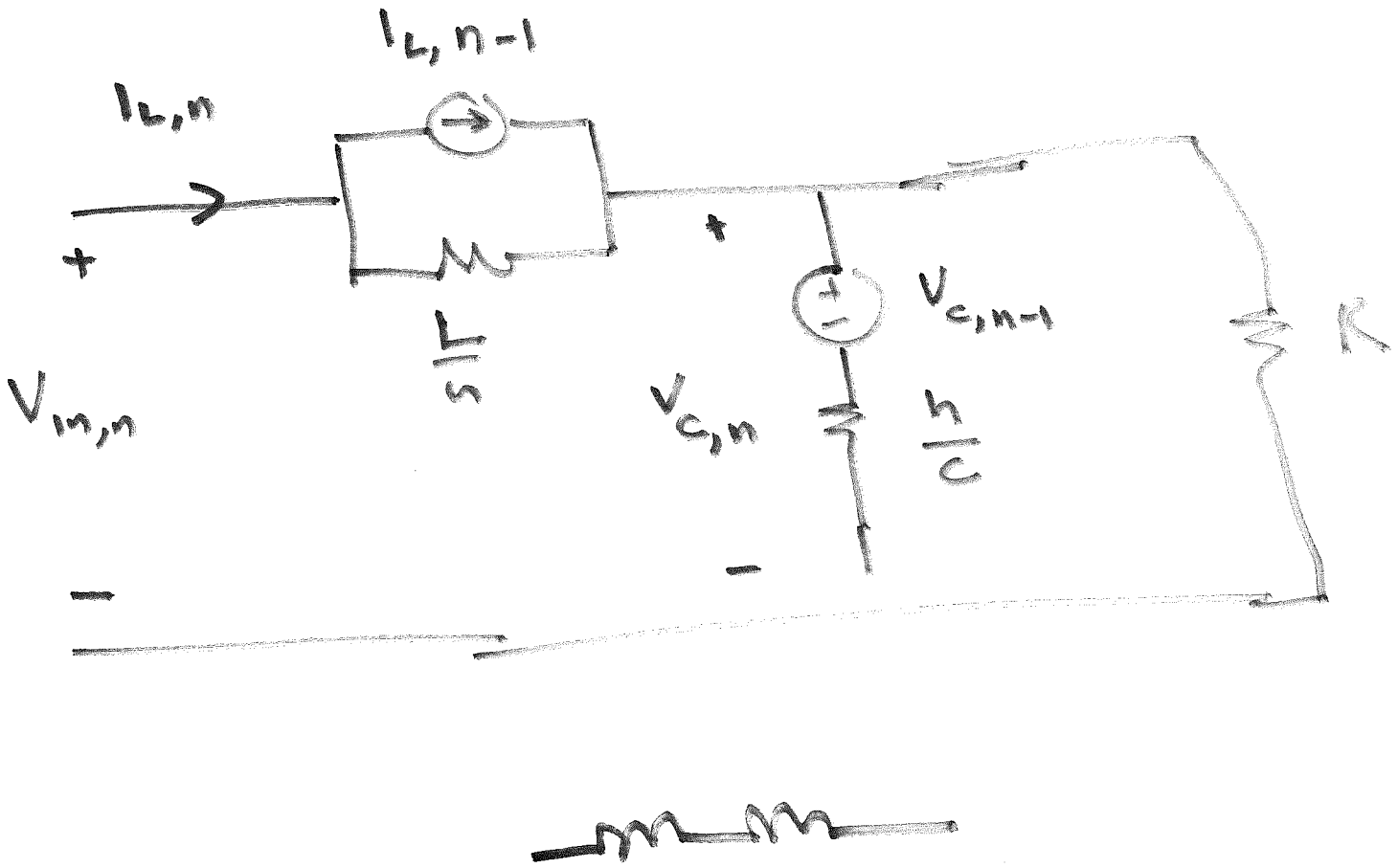
$$= V_{c,n-1} + \frac{1}{C}\left(I_{L,n} - \frac{V_{c,n}}{R}\right)h$$

$$= V_{c,n-1} + \frac{h}{C} I_{c,n}$$





Resistor-Companion Simulation [1]



State Variable Based Simulation

- In general

$$\frac{d\mathbf{x}}{dt} = p\mathbf{x} = \mathrm{f}(\mathbf{x},\mathbf{u}) \qquad \frac{d\mathbf{x}}{dt} = p\mathbf{x} = \mathrm{f}(\mathbf{x},\mathbf{u},t) \qquad \frac{d\mathbf{x}}{dt} = p\mathbf{x} = \mathrm{f}(\mathbf{x},t)$$

$$\mathbf{y} = \mathrm{g}(\mathbf{x},\mathbf{u}) \qquad \mathbf{y} = \mathrm{g}(\mathbf{x},\mathbf{u},t) \qquad \mathbf{y} = \mathrm{g}(\mathbf{x},t)$$

- For linear time-invariant systems

$$\frac{d\mathbf{x}}{dt} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$

$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}$$

- For linear time-varying systems

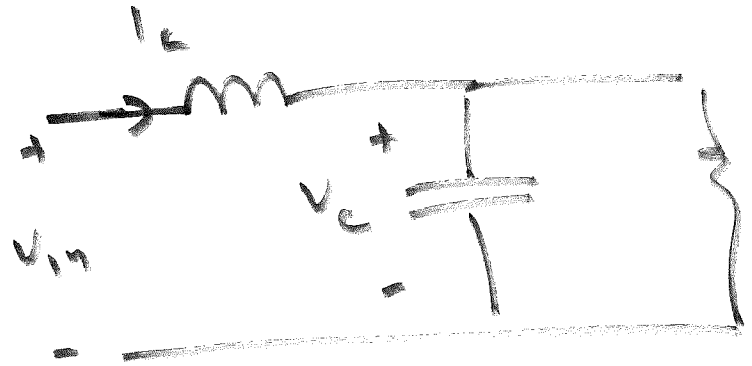
$$\frac{d\mathbf{x}}{dt} = \mathbf{A}(t)\mathbf{x} + \mathbf{B}(t)\mathbf{u}$$

$$\mathbf{y} = \mathbf{C}(t)\mathbf{x} + \mathbf{D}(t)\mathbf{u}$$

State Variables

- Definition: $\mathbf{x} \in R^n$ is a minimum set of variables that given their value at $t = t_0$ and the system inputs as a function of time can be used to predict system performance for $t > t_0$.

- Example:



$$L\frac{dI_L}{dt} = V_{in} - V_C$$

$$\frac{dI_L}{dt} = \frac{1}{L}(V_{in} - V_C)$$

$$C\frac{dV_C}{dt} = I_L - \frac{V_C}{R}$$

$$\frac{dV_C}{dt} = \frac{1}{C}\left(I_L - \frac{V_C}{R}\right)$$

State Variables

$$x = \begin{bmatrix} i_L \\ v_c \end{bmatrix} \qquad u = [V_{in}]$$

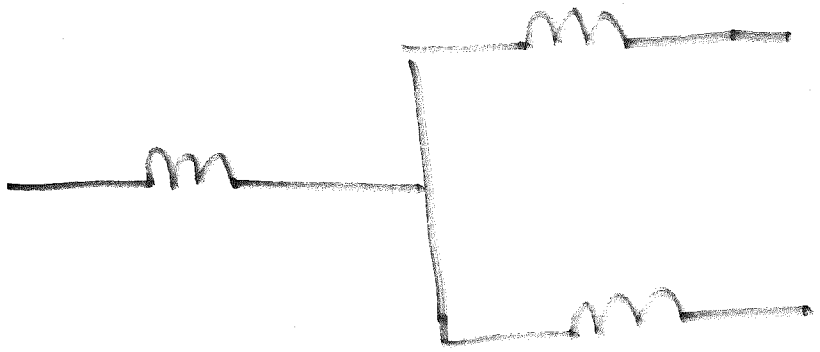
$$\dot{x} = \begin{bmatrix} 0 & -\frac{1}{L} \\ \frac{1}{C} & -\frac{1}{RC} \end{bmatrix} x + \begin{bmatrix} \frac{1}{L} \\ 0 \end{bmatrix} u = f(x,u)$$

$$\lambda_L = L i_L \qquad \frac{d\lambda_L}{dt} = V_{in} - v_c = V_{in} - \frac{Q_c}{C}$$

$$Q_c = C v_c \qquad \frac{dQ_c}{dt} = \frac{\lambda_L}{L} - \frac{Q_c}{RC}$$

Uniqueness of State Variables

$$X = \begin{bmatrix} \frac{d\lambda_L}{dt} \\ \frac{dv_c}{dt} \end{bmatrix} \qquad pX = \begin{bmatrix} 0 & -\frac{1}{C} \\ \frac{1}{L} & -\frac{1}{RC} \end{bmatrix} X + \begin{bmatrix} 1 \\ 0 \end{bmatrix} v$$



Existence and Solution of ODE's

- Picard-Kubderköf Theorem
- Suppose

$$p\mathbf{x} = \mathrm{f}(\mathbf{x}, t)$$
$$\mathbf{x}(t_0) = \mathbf{x}_0 \qquad (1)$$

- If f(**x**,t) is Lipschitz continuous, then there exists a solution to (1) for t_0-ε < t < t_0+ε

Lipschitz Continuity

- A function is Lipschitz continuous if

$$\left\|\mathrm{f}(\mathbf{x}_2)-\mathrm{f}\left(\mathbf{x}_1\right)\right\|<K\left\|\mathbf{x}_2-\mathbf{x}_1\right\|$$

- Example 1

$$\dot{x} = Ax + Bu$$

$$\dot{x}_2 = Ax_2 + Bu$$

$$\dot{x}_1 = Ax_1 + Bu$$

$$\|\dot{x}_2 - \dot{x}_1\| = \|A(x_2 - x_1)\|$$

$$\|A\| = \max \frac{\|Ax\|}{\|x\|} = \sqrt{\max \lambda(A^T A)}$$

$$\|Ax\| \leq \|A\| \|x\|$$

Lipschitz Continuity

$$\|A(x_2 - x_1)\| \leq \underbrace{\|A\|}_{K} \|x_2 - x_1\|$$

$$f(x) = x^2$$

$$f(x_2) - f(x_1) = x_2^2 - x_1^2 = (x_2 + x_1)(x_2 - x_1)$$

$$\|f(x_2) - f(x_1)\| = \underbrace{\|x_2 + x_1\|}\, \|x_2 - x_1\|$$

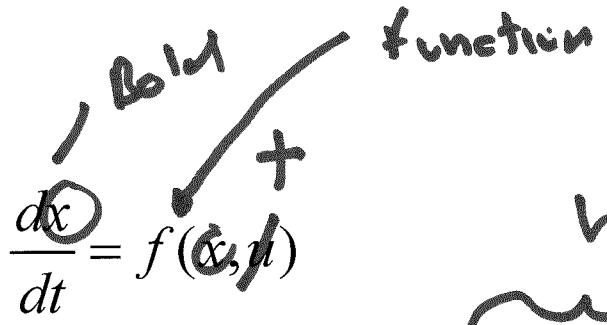
$\Rightarrow$ Not Lipschitz continuous

State Model Solution Methods

- Explicit
  - Forward Euler
  - Runga-Kutta
  - Many Others
- Implicit
  - Backward Euler
  - Trapezoidal
  - Many Others

Forward Euler

- Consider



$$\frac{dx}{dt} = f(x,t)$$

$$x_n = x_{n-1} + \left.\frac{dx}{dt}\right|_{x_{n-1}} (t_n - t_{n-1})$$

$$= x_{n-1} + F(x_{n-1}, t_{n-1})\, h$$

$$x_{n+1} = x_n + F(x_n, t_n)\, h$$

Quick Example

- Suppose we have

$$px_1 = -x_1 + 0.1x_2^2 + u_1$$
$$px_2 = -x_2 + 0.1x_1$$

- And that

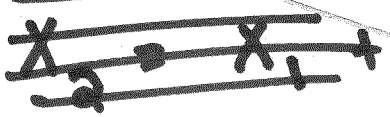
$$x_1(t_1) = 5$$
$$x_2(t_1) = 1$$
$$u_1(t_1) = 0$$

At time t_1

$px_1 = -5 + 0.1(1)^2 + 0$

$= -4.9$

$px_2 = -1 + 0.1(5) = -0.5$



meet!

- Find $x(t_2)$ where $t_2 = t_1 + h$; $h = 0.1$ s

simulation

$$x(t_2) = x(t_1) + h\, px(t_1)$$
$$= \begin{bmatrix} 5 \\ 1 \end{bmatrix} + 0.1 \begin{bmatrix} -4.9 \\ -0.5 \end{bmatrix} = \begin{bmatrix} 4.51 \\ 0.95 \end{bmatrix}$$

Quick Example (Continued)

Observations from Euler's Method

- To predict a future value of state, we needed to find the present value of the time derivative of the state variables, based on what we know – that is the present value of state variables and present value of input variables.

- This is true of all other integration algorithms as well

The Runge-Kutta Algorithm

- A practical explicit algorithm for

$$\frac{d\mathbf{x}}{dt} = \mathrm{f}(\mathbf{x}, t)$$

- The fourth-order implementation (RK4)

order ↓ $p+1$

error $\leq c\, h^{p+1}$

$$\mathbf{k}_1 = \mathrm{f}(\mathbf{x}_n, t_n)$$

$$\mathbf{k}_2 = \mathrm{f}\left(\mathbf{x}_n + \frac{1}{2}h\mathbf{k}_1, t_n + \frac{1}{2}h\right)$$

$$\mathbf{k}_3 = \mathrm{f}\left(\mathbf{x}_n + \frac{1}{2}h\mathbf{k}_2, t_n + \frac{1}{2}h\right)$$

$$\mathbf{k}_4 = \mathrm{f}\left(\mathbf{x}_n + h\mathbf{k}_3, t_n + h\right)$$

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \frac{h}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4)$$

$$t_{n+1} = t_n + h$$

Backward Euler

$$x_n = x_{n-1} + f(x_n, t_n)\underbrace{(t_n - t_{n-1})}_{h}$$

$f(x_{n-1}, t_{n-1})$

→ numerical stability

→ implicit.

Trapezoidal

Forward Euler: $\dot{x} = f(x_{n-1}, t_{n-1})$

Backward Euler: $\dot{x} = f(x_n, t_n)$

Trap $\dot{x} = \frac{1}{2}\left[f(x_{n-1}, t_{n-1}) + f(x_n, t_n)\right]$

$$x_n = x_{n-1} + \frac{h}{2}\left[f(x_{n-1}, t_{n-1}) + f(x_n, t_n)\right]$$

Trapezoidal Predictor-Corrector Method

- Again consider $\frac{d\mathbf{x}}{dt} = \mathrm{f}(\mathbf{x}, t)$
- Algorithm
  - Step 1- Initial Guess

$$g = 1$$

$$t_n = t_{n-1} + h$$

$$\breve{\mathbf{x}}_{n,new} = \mathbf{x}_{n-1} + h\,\mathrm{f}(\mathbf{x}_{n-1}, t_{n-1})$$

  - Step 2 – Iterate

$$\breve{\mathbf{x}}_{n,old} = \breve{\mathbf{x}}_{n,new}$$

$$\breve{\mathbf{x}}_{n,new} = \mathbf{x}_{n-1} + \frac{h}{2}\left(\mathrm{f}(\mathbf{x}_{n-1}, t_{n-1}) + \mathrm{f}(\breve{\mathbf{x}}_{n,old}, t_n)\right)$$

$$g = g + 1$$

until $g=G$ or error criteria satisfied

- Step 3 – Finish step

$$\mathbf{x}_n = \breve{\mathbf{x}}_{n,new}$$

Trapezoidal Predictor-Corrector Method

- Error Criteria

$$x_d = |\tilde{x}_{n,old} - \tilde{x}_{n,new}|$$ ← element wise ||

$$x_s = |\tilde{x}_{n,old} + \tilde{x}_{n,new}|$$

$$r = \max(x_d - \varepsilon_{rel} x_s) < 0$$

ε_{rel}: relative error criteria

$$a = \|x_d\|_\infty < \varepsilon_{abs}$$

ε_{abs}: absolute error criteria

Convergence criteria = r or a

Simulation Vocabulary

- Suppose

$$\frac{d\mathbf{x}}{dt} = \mathrm{f}(\mathbf{x}, t)$$

$$h = t_n - t_{n-1}$$

$$\mathbf{x}_n = \mathbf{x}_{n-1} + h\,\mathrm{g}(\mathbf{x}_{n-1}, t_{n-1}, h, \mathrm{f})$$

- Local Truncation Error

$$\mathbf{e}_n = \mathbf{x}(t_n) - \left(\mathbf{x}(t_n - 1) + h\,\mathrm{g}(\mathbf{x}(t_n - 1), t_{n-1}, h, \mathrm{f})\right)$$

x_n | if we had perfect knowledge of $x_{n-1} = x(t_{n-1})$

t_{n-1} t_{n-1}

exact solution

Simulation Vocabulary

- Method is consistent if local truncation error is $o(h)$
  - $o(h)$ means for every $\varepsilon > 0$ there exists an H such that $\|e_n\| < \varepsilon h$ for all $h < H$

- A method is order p if the local truncation error is $o(h^{p+1})$

Example: Forward Euler

- Consider first order ODE

$$\frac{dx}{dt} = ax + bu$$

$$x(0) = 0$$

u constant

- Analytical solution

$$x = -\frac{bu}{a}\left(1 - e^{at}\right)$$

Example: Forward Euler (Continued …)

$$x_n = x_{n-1} + h f(x_{n-1}, t_{n-1})$$

$$= x_{n-1} + h \left[a x_{n-1} + bu \right]$$

$$x_n = [1 + ah] x_{n-1} + hbu$$

numerically stable iff

$$-1 < 1 + ah < 1$$

$$-2 < ah < 0$$

$$-\frac{2}{a} > h$$

$$\boxed{h < -\frac{2}{a}}$$

Example: Forward Euler (Continued …)

$$e_n = X(t_n) - X_n \Big|_{\text{if we knew } X(t_{n-1})}$$

$$= -\frac{bu}{a}\left(1 - e^{at_n}\right)$$

$$\rightarrow X(t_{n-1}) + h\,F\left(X(t_{n-1}), t_{n-1}\right)$$

$$- \left[-\frac{bu}{a}\left(1 - e^{a(t_n - h)}\right) + h\left[a\left(-\frac{bu}{a}\left(1 - e^{a(t_n-h)}\right)\right) + bu \right] \right]$$

$$= -\frac{bu}{a} + \frac{bu}{a} e^{at_n}$$

$$+ \frac{bu}{a} - \frac{bu}{a} e^{a(t_n - h)} + ha\frac{bu}{a}\left(1 - e^{a(t_n-h)}\right)$$

$$= \frac{bu}{a} e^{at_n}\left[1 - e^{-ah} - hae^{-ah}\right] \quad * -hbu$$

Example: Forward Euler (Continued …)

$$e^x = 1 + x + \tfrac{1}{2}x^2 + H$$

$$e^{-ah} = 1 - ah + \tfrac{1}{2}(ah)^2 + H$$

$$e_n = \frac{bu}{a} e^{ath}\left[1 - \left(1 - ah + \tfrac{1}{2}(ah)^2 + H\right) - ha\left(1 - ah + \tfrac{1}{2}(ah)^2 + H\right)\right]$$

$$= \frac{bu}{a} e^{ath}\left[1 - 1 + ah - \tfrac{1}{2}(ah)^2 - H - ha + (ah)^2 - \tfrac{1}{2}(ah)^3 - ahH\right]$$

$$= \underbrace{\frac{bu}{a} e^{ath} \tfrac{1}{2} a^2}\, h^2 + H$$

↳ 1st order

Example: Trapezoidal

$$x_n = x_{n-1} + \frac{h}{2}\left[f(x_{n-1}, t_{n-1}) + f(x_n, t_n)\right]$$

$$f = ax + bu$$

$$x_n = x_{n-1} + \frac{h}{2}\left[a x_{n-1} + bu + a x_n + bu\right]$$

$$x_n\left[1 - \frac{ah}{2}\right] = \left[1 + \frac{ah}{2}\right] x_{n-1} + hbu$$

$$x_n = \frac{\left[1 + \frac{ah}{2}\right]}{\left[1 - \frac{ah}{2}\right]} x_{n-1} + \frac{hb}{1 - \frac{ah}{2}} u$$

Example: Trapezoidal (Continued …)

We need

$$-1 \leq \frac{1+\frac{ah}{2}}{1-\frac{ah}{2}} \leq 1$$

$$-1+\frac{ah}{2} \leq 1+\frac{ah}{2} \leq 1-\frac{ah}{2}$$

$$-1 \leq 1+[\text{illegible}] \leq 1-ah$$

$\Rightarrow$ any h will work if the system is stable ($a<0$).

Example: Trapezoidal (Continued ...)

Local Truncation Error

$$e_n = -\frac{1}{12} b U a^2 e^{a t_n} h^3$$

h^3 → 2nd order

Example: Trapezoidal (Continued ...)

Simulation Vocabulary

- Global Truncation Error

$$e_n = \mathbf{x}(t_n) - \mathbf{x}_n$$

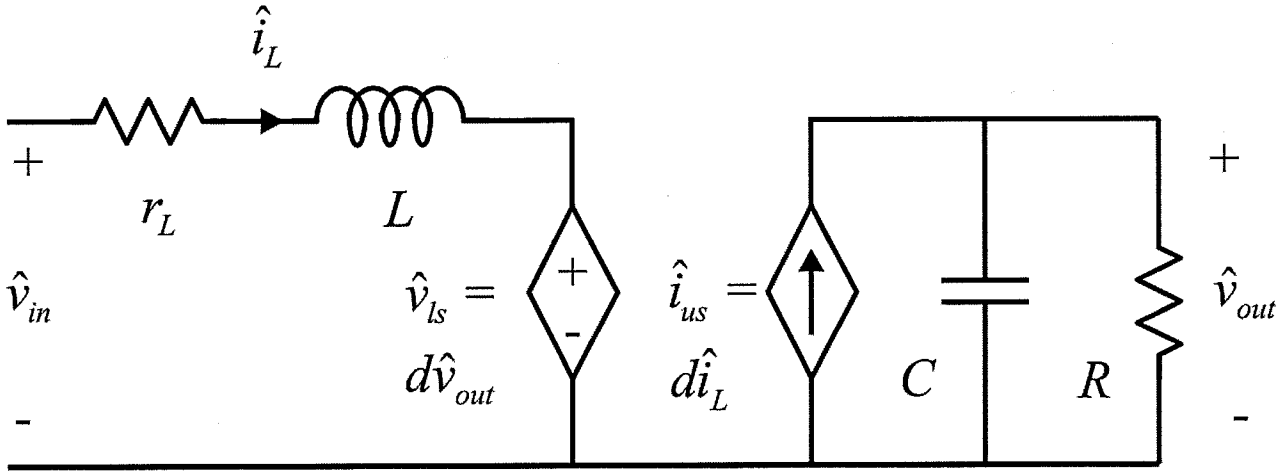
↳ For local error X_n is X_n calculated based on $X(t_{n-1})$.
→ For global error, X_n is based on X_{n-1}

- A method is convergent if

$$\lim_{h \to 0} \max_n \|e_n\| = 0$$

Example System (Boost Converter)

- Lets consider an average-value model of a dc-dc converter



$$\hat{v}_{ls} = d\hat{v}_{out}$$

$$\hat{i}_{us} = d\hat{i}_L$$

$$\hat{i}_{out} = \frac{\hat{v}_0}{R}$$

$$p\hat{i}_L = \frac{\hat{v}_{in} - \hat{v}_{ls} - r_L\hat{i}_L}{L}$$

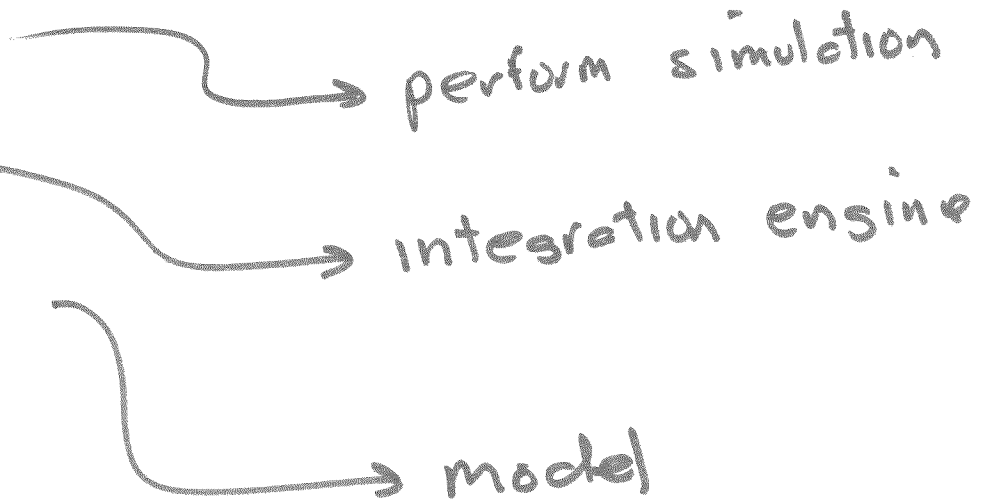
$$p\hat{v}_{out} = \frac{\hat{i}_{us} - \hat{i}_{out}}{C}$$

Example System (Boost Converter)

- System Parameters
  - $R = 10\ \Omega$
  - $L = 5$ mH
  - $r_L = 0.5\ \Omega$
  - $C = 1000$ uF
  - $v_{in} = 300$ V
- The converter is in the steady-state. Then the duty cycle steps from 0.5 to 0.7 at t = 20 ms.

Matlab Implementation

- msim_dcdc.m
- odefsrk.m
- mavm_dcdc.m

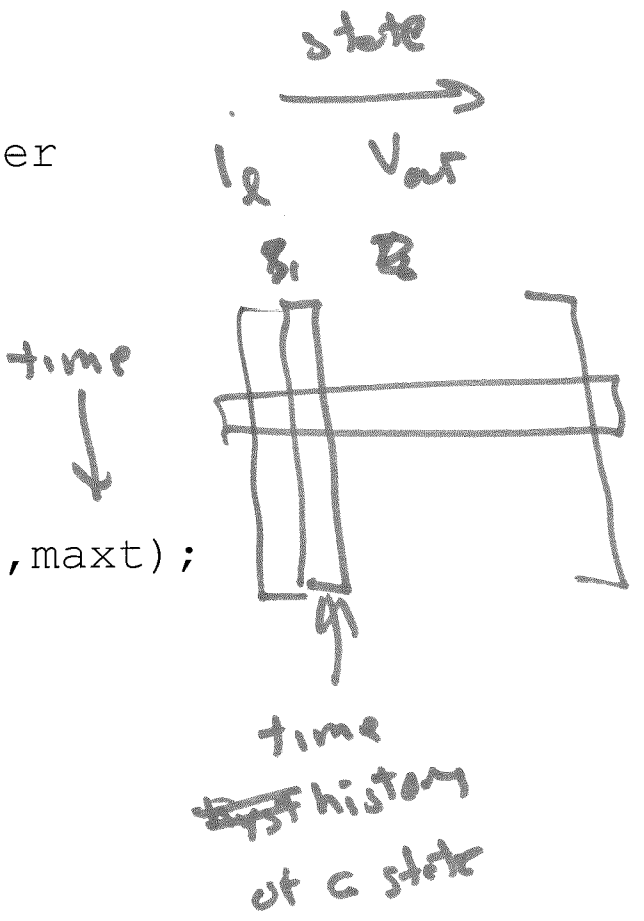


msim_dcdc.m

```
% This script sets parameter and simulates a
% simple matlab average value dc-dc (boost) converter using mavm_dcdc
%
% Written by S.D. Sudhoff
%   School of Electrical and Computer Engineering
%   1285 Electrical Engineering Building
%   West Lafayette, IN 47907-1285
%   Phone: 765-494-3246
%   Fax: 765-494-0676
%   E-mail: sudhoff@ecn.purdue.edu

% model parameters
P.L=5e-3;                    % inductor inductance (H)
P.rL=0.5;                    % inductor resistance (Ohms)
P.C=1000e-6;                 % capacitor capacitance (F)
P.Rload=10.0;                % load resistance (Ohms)
P.di=0.5;                    % initial duty cycle
P.df=0.7;                    % final duty cycle
P.tstep=0.02;                % time for duty cycle step
P.vin=300;                   % input voltage(V)
```

msim_dcdc.m

```
% compute intial conditions
vouti=P.vin/(P.rL/(P.Rload*P.di)+P.di);      % initial output
voltage (V)
ili=vouti/(P.di*P.Rload);                    % initial inductor
current (A)
xic=[ili vouti];

% time values where we want the answer
tvals=linspace(0,0.1,200);

% maximum time step
maxt=2e-5;

% peform the simulation
[t,x]=odefsrk(@mavm_dcdc,tvals,xic,P,maxt);

% grab the waveforms
iin=x(:,1);
vout=x(:,2);
```



msim_dcdc.m

```
% plot the voltage
figure(1)
plot(tvals,vout)
xlabel('t, s');
ylabel('v_{out}, A');
title('Output Voltage Versus Time');
axis([0 0.1 0 550]);
grid on

% plot the current
figure(2)
plot(tvals,iin)
xlabel('t, s');
ylabel('i_{in}, A');
title('Input Current Versus Time');
axis([0 0.1 0 120]);
grid on
```

odefsrk.m

```
function [t,y]=odefsrk(fhandle,tspan,yic,par,maxt)
% This routine solves a ordinary differential equation
% using a 4th order Runga-Kutta method.
%
% [t,y] = odefsrk(fhandle,tspan,yic,par,maxt);
%
% Inputs:
%
% fhandle  = a handle to the function whose output is the time derivative
%            of the system model.  The inputs to this function are time,
%            state, and parameter vales.
% tspan    = a vector whose elements describe at which point in time the
%            solution is sought
% yic      = a vector which describes the initial condition of the system
%            being simulated
% par      = a structure which contains data or parameters needed to
%            evaluate the time derivative of the state variables
% maxt     = the maximum allowed time step
```

odefsrk.m

```
%
% Outputs:
%
% t         = a vector of times at which the state vector has been found
% y         = a matrix wherein each row contains the state vector at a
%             given time.  Each column is the time history of a particular
%             state
%
% Internal:
%
% Nt        = Number of points at which to calculate the state vector
% Ns        = Number of state variables
% i         = index variable for time
% deltat    = time span between current interval being simulated (s)
% nsteps    = number of integration steps required on interval
% h         = integration time step (s)
```

odefsrk.m

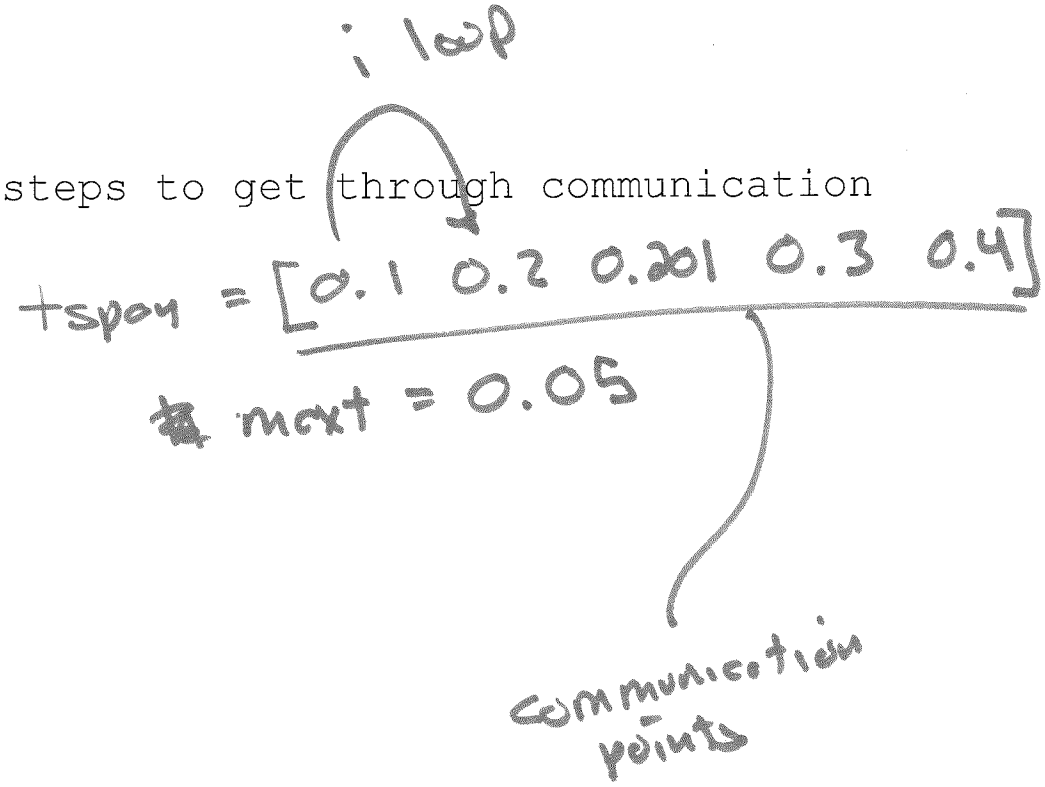
```
% ho2       = half of integration time step (s)
% y1        = state vector at beginning of integration step
% t1        = time at the beginning of the integration step (s)
% k1        = time derivative of state vector at beginning of integration
%             step (s)
% y2        = 1st est. of state vector in center of integration step
% k2        = time derivative of y2 at center of integration step
% y3        = 2nd est. of state vector in center of integration step
% k3        = time derivative of y3 at center of integration step
% y4        = estimate of state vector at end of integration step
% k4        = time derivative of state vector at end of integration step
% k         = weighted average value of time derivative of integration
%             step
%
% Written by S.D. Sudhoff
%   School of Electrical and Computer Engineering
%   1285 Electrical Engineering Building
%   West Lafayette, IN 47907-1285
%   Phone: 765-494-3246
%   Fax: 765-494-0676
%   E-mail: sudhoff@ecn.purdue.edu
```

odefsrk.m

```
% determine number of times at which state vector is recorded
Nt=length(tspan);

% determine the nubmer of states
Ns=length(yic);

% assign the time vector, and initialize y matrix
t=tspan;
y=zeros(Nt,Ns);

% first value row of y matrix is determined in accordance with
% initial condition
y(1,:)=yic;

% iterate one communication interval at a time.  A given
communication
% interval may require sevaral time steps
for i=2:Nt;
```

odefsrk.m

```
% find integration information for this communication interval
  deltat=tspan(i)-tspan(i-1);
  nsteps=ceil(deltat/maxt);
  h=deltat/nsteps;
  ho2=0.5*h;
  y1=y(i-1,:)';
  t1=tspan(i-1);

  % now perform enough integration steps to get through communication
  % interval
  for j=1:nsteps

     % perform Runga-Kutta update
     t2=t1+ho2;
     t3=t1+h;

     k1=fhandle(t1,y1,par);

     y2=y1+ho2*k1;
     k2=fhandle(t2,y2,par);
```



odefsrk.m

```
        y3=y1+ho2*k2;
        k3=fhandle(t2,y3,par);

        y4=y1+h*k3;
        k4=fhandle(t3,y4,par);

        k=(k1+2.0*k2+2.0*k3+k4)/6.0;
        y1=y1+k*h;

        t1=t1+h;

    end

    % fill in the y-matrix
    y(i,:)=y1';

end
```

mavm_dcdc

```
function [varargout]=mavm_dcdc(t,x,P)
% This routine contains the dynamics of an simple dc-dc (boost) converter
% model.
%
% [px] = mavm_dcdc(t,x,P);
% [px,vls,ius,iout] = mavm_dcdc(t,x,P);
%
% Inputs:
% t         = time (s)
% x         = state vector
%  x(1)     = fast average of inductor current (A)
%  x(2)     = fast average of output voltage (V)
% P         = structure with parameters
%  P.L      = inductor inductance (H)
%  P.rL     = inductor series resistance (Ohms)
%  P.C      = capacitor capacitance (F)
%  P.Rload  = load resistance (Ohms)
%  P.vin    = input voltage (V)
%  P.di     = initial duty cycle (upper switch on / total period)
%  P.df     = final duty cycle
%  P.tstop  = time of step in duty cycle (s)
%
```

mavm_dcdc

```
% Outputs:
% px        = time derivative of state vector
%  px(1)    = time derivative of fast average of inductor current (A)
%  px(2)    = time derivative of fast average of output voltage (V)
% vls       = fast average lower switch voltage (V)
% ius       = fast average current out ot upper switch (A)
% iout      = output current (A)
%
% Internal:
% d         = duty cycle
% pil       = time derivative of il (A/s)
% pvout     = time derivative of vout (V/s)
% il        = fast average inductor current (A)
% vout      = fast average output voltage (V)
%
% Written by S.D. Sudhoff
%   School of Electrical and Computer Engineering
%   1285 Electrical Engineering Building
%   West Lafayette, IN 47907-1285
%   Phone: 765-494-3246
%   Fax: 765-494-0676
%   E-mail: sudhoff@ecn.purdue.edu
```

mavm_dcdc

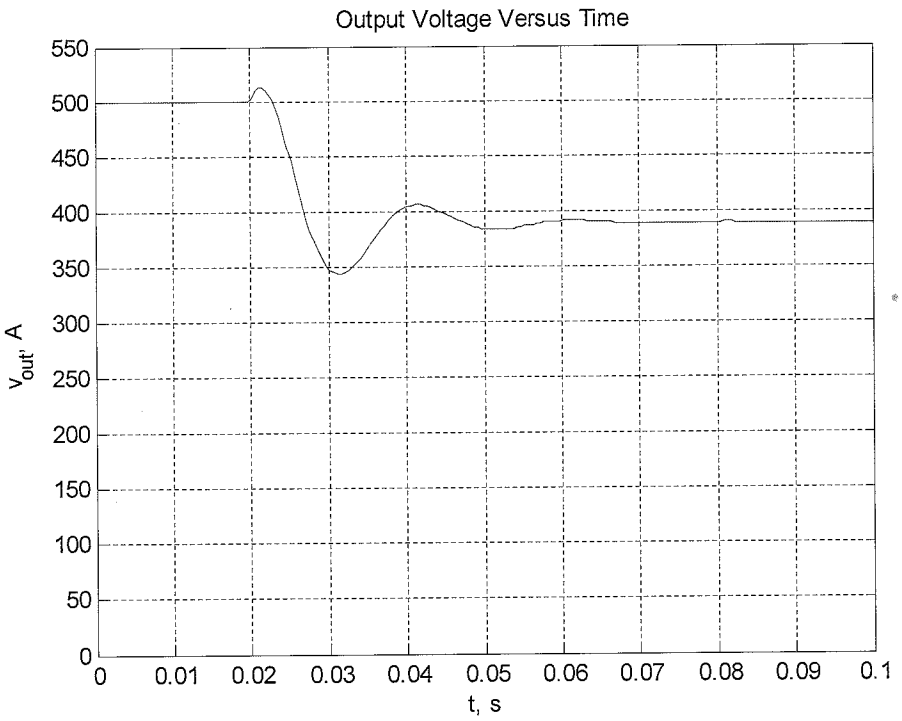
```
% decompose state vector
il=x(1);
vout=x(2);

% compute the duty cycle
if (t<P.tstep)
   d=P.di;
else
   d=P.df;
end

% variables of interest
vls=d*vout;
ius=d*il;
iout=vout/P.Rload;
```

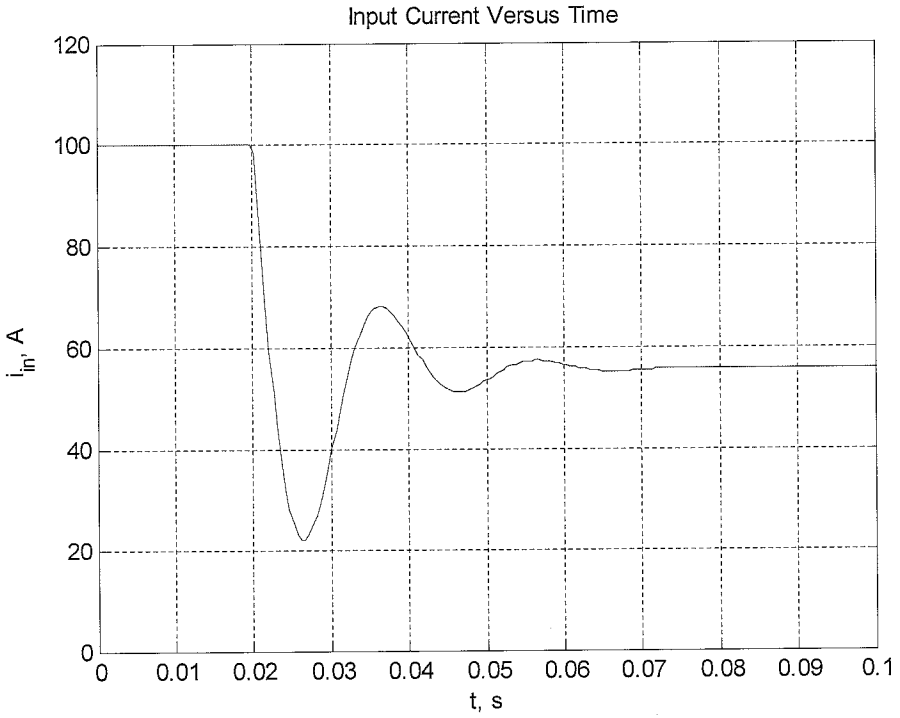
Sorted

mavm_dcdc

```
% compute derivatives of state
pil=(P.vin-P.rL*il-vls)/P.L;
pvout=(ius-iout)/P.C;

% pack the state vector
px=[pil pvout]';

% output variables
if nargout==1
   varargout={px};
else
   varargout={px,vls,ius,iout};
end

end
```

Results


Output Voltage Versus Time
550
500
450
400
350
300
250
200
150
100
50
0
v_{out}, A
0
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0.09
0.1
t, s


Input Current Versus Time
120
100
80
60
40
20
0
i_{in}, A
0
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0.09
0.1
t, s

ECE61016 Power Electronics Converters and Systems

Time-Domain Simulation

Homework 1

S.D. Sudhoff

Fall 2016

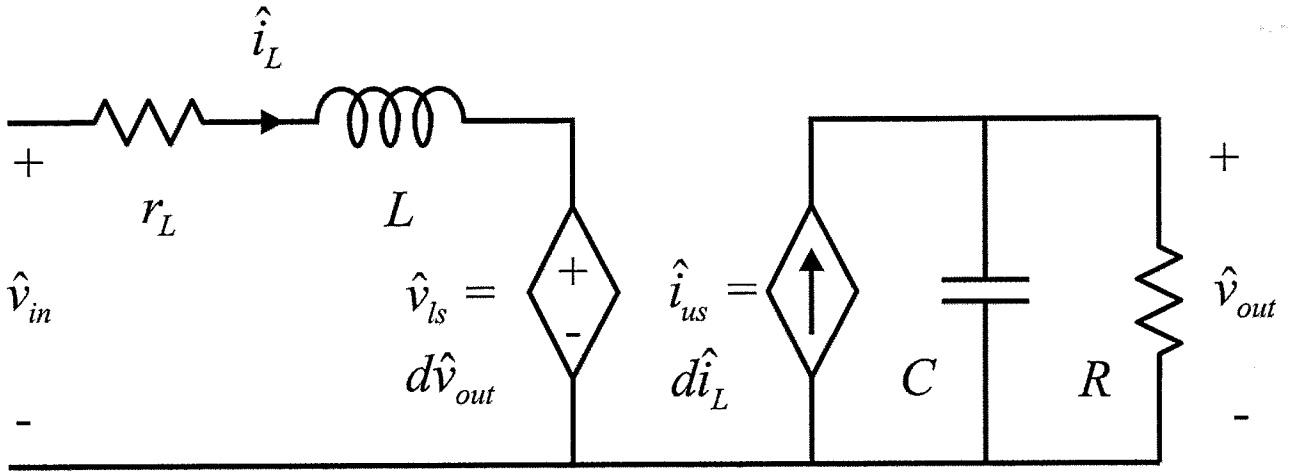
Aspects of Assignment

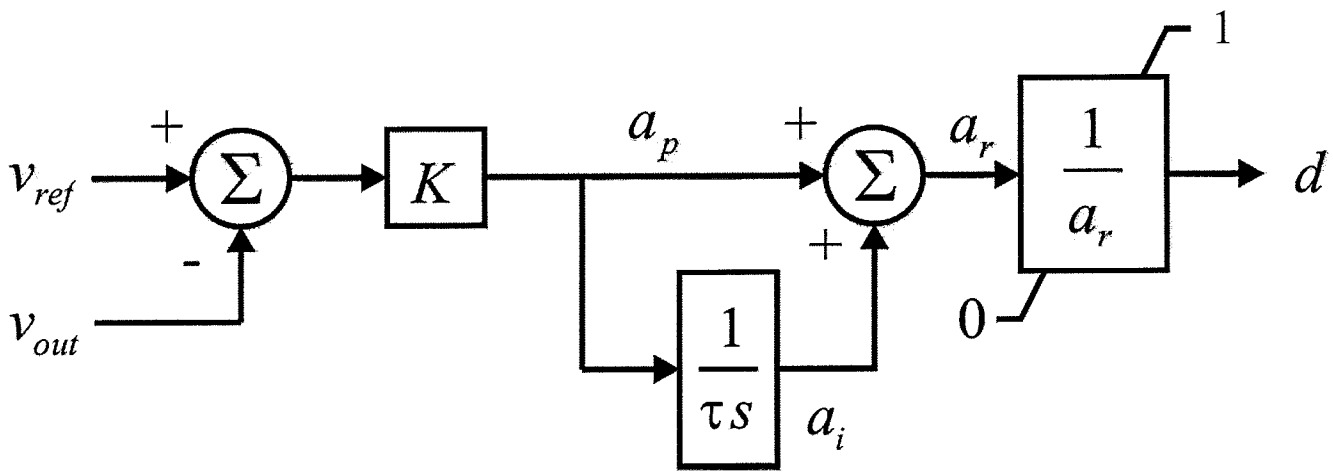
- Problem 1: Add a controller to our dc/dc converter
- Problem 2: Code a trapezoidal predictor corrector routine and use in our system.
- Problem 2+: If you have already had ECE633, also work the following. Consider the scalar system

$$px = ax + bu$$

Assuming $a<0$, and u is constant, find a bound on h when using the 4^{th} order Runga-Kutta algorithm. It should be in terms of a.

DCDC Converter and Control





$L = 5\,\text{mH}$

$r_L = 100\,\text{m}\Omega$

$C = 100\,\mu\text{F}$

$K = 10^{-3}\,\text{V}^{-1}$

$\tau = 10^{-3}\,\text{s}$

Problem 1

- Given
  - odefsrk.m (integration routine)
  - regdcdc_rksim.m (processing script)
- Write regulated dc/dc converter model
  - regdcdc.m
- Deliverables
  - regdcdc.m (in printout) documentation
  - Plots of run including inductor current, output voltage, duty cycle (in printout).
  - Do not change scales for plots. If you need to do this, your answer is incorrect!

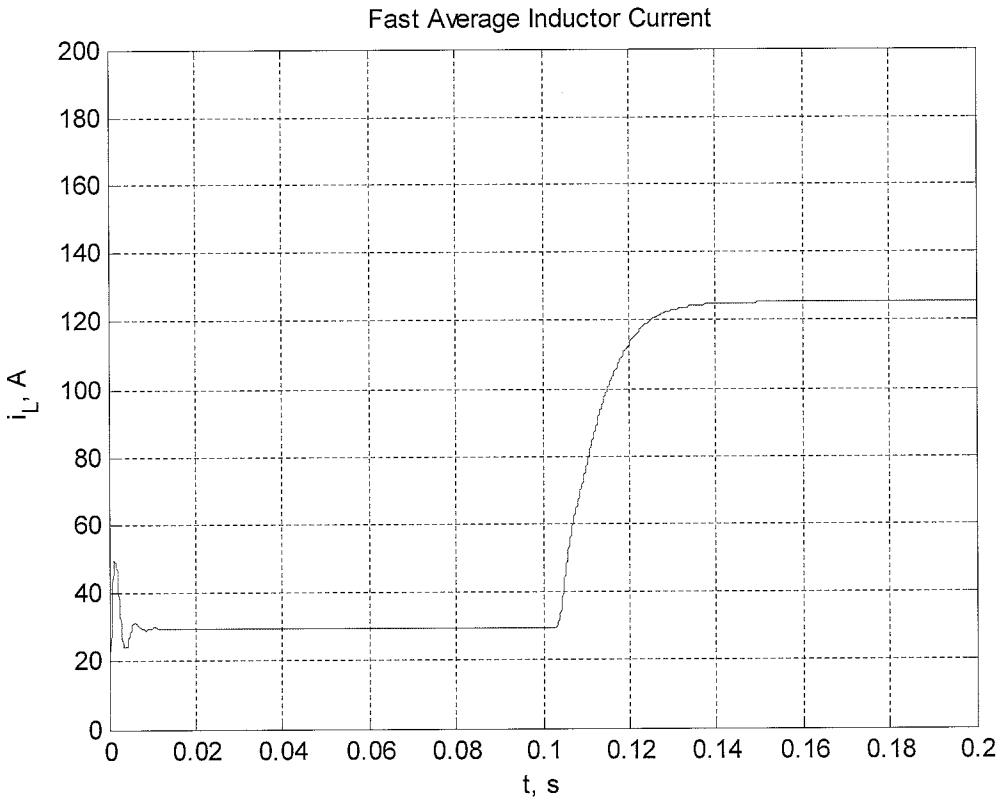
regdcdc.m

```
function [varargout]=regdcdc(t,x,P)
% This routine contains the dynamics of an simple regulated dc/dc converter
% model
%
% [px] = regdcdc(t,x,P);
% [il,vout,vls,ius,iout,d] = regdcdc(t,x,P);
%
% Inputs:
%
% t         = time (s)
% x         = state vector
%  x(1)     = fast average of inductor current (A)
%  x(2)     = fast average of output voltage (V)
%  x(3)     = fast average of integrator output ()
% P         = structure with parameters
%  P.L      = inductor inductance (H)
%  P.rL     = inductor series resistance (Ohms)
%  P.C      = capacitor capacitance (F)
%  P.R      = load resistance (Ohms)
```

regdcdc.m (continued)

```
%  P.vin   = input voltage (V)
%  P.vref0 = initial reference voltage (V)
%  P.vref1 = final reference voltage (V)
%  P.tref  = time for switch in reference voltage (s)
%  P.K     = controller gain
%  P.tau   = controller time constant (s)
%
% Outputs:
%
% px       = time derivative of state vector
%  px(1)   = time derivative of fast average of inductor current (A)
%  px(2)   = time derivative of fast average of output voltage (V)
%  il      = fast average inductor current (A)
%  vout    = fast average output voltage (V)
%  vls     = fast average lower switch voltage (V)
%  ius     = fast average current out of upper switch (A)
%  d       = duty cycle
```

Problem 1 – Sample Run



Problem 2

- Given
  - regdcdc_tpcsim.m (simulation script)
  - regdcdc.m (model dynamics from problem 1)
- Write
  - odetpc.m (trapezoidal predictor-corrector algorithm)
- Deliverables
  - printout of odetpc.m
  - printout of study (inductor current, output voltage, duty cycle)

odetpc.m

```
function [t,y]=odetpc(fhandle,tspan,yic,par,SP)
% This routine solves a ordinary differential equation
% using a trapezoidal predictor corrector method.
%
% [t,y] = odetpc(fhandle,tspan,yic,par,maxt);
%
% Inputs:
%
% fhandle  = a handle to the function whose output is the time derivative
%            of the system model.  The inputs to this function are time,
%            state, and parameter vales.
% tspan    = a vector whose elements describe at which point in time the
%            solution is sought
% yic      = a vector which describes the initial condition of the system
%            being simulated
% par      = a structure which cointains data or parameters needed to
%            evaluate the time derivative of the state variables
% SP       = structure of simulation algorithm parameters
%  SP.maxt = the maximum allowed time step (s)
%  SP.maxit= maximum allowed iterations
%  SP.maxre= maximum relative error in state variable
%  SP.maxae= maximum absolute error in state variable
%
```

odetpc.m continued

```
% Outputs:
%
% t         = a vector of times at which the state vector has been found
% y         = a matrix wherein each row contains the state vector at a
%             given time.  Each column is the time history of a particular
%             state
```

Simulink Implementation

- savm_dcdc_setup.m → sets parameters (script)
- savm_dcdc.slx → model
- savm_dcdc_plot.m → plotting routine for every component function
- savm_dcdc_pp.m → post processing file (script)
- savm_dcdc_study.m → everything (script)

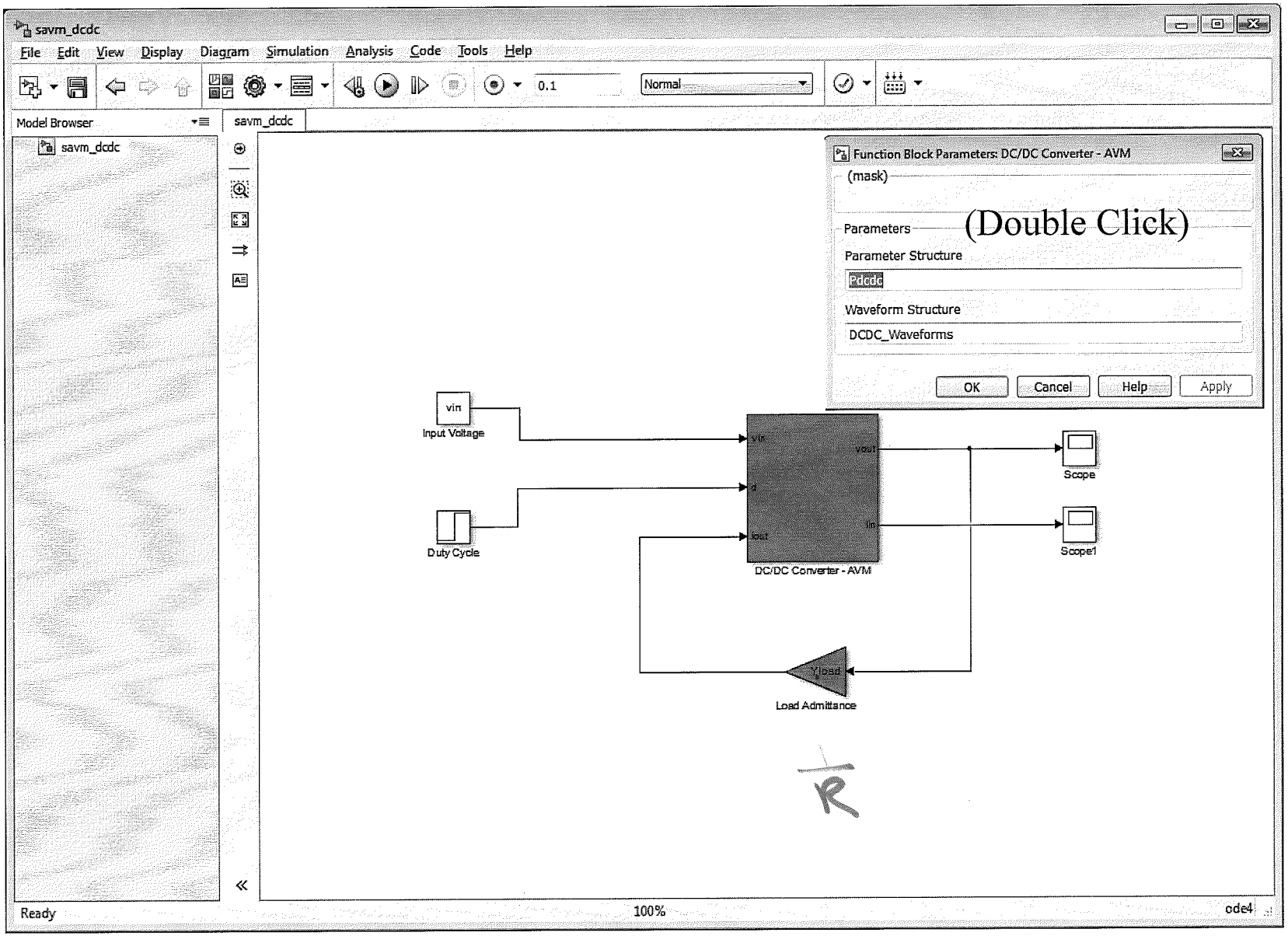
savm_dcdc_setup

```
% This script set parameter and performs intial calculations for
% simple simulink average value dc/dc converter simulation savm_dcdc

% Written by S.D. Sudhoff
%   School of Electrical and Computer Engineering
%   1285 Electrical Engineering Building
%   West Lafayette, IN 47907-1285
%   Phone: 765-494-3246
%   Fax: 765-494-0676
%   E-mail: sudhoff@ecn.purdue.edu

% input source
vin=300;                                    % input voltage (V)
```

savm_dcdc_setup

```
% duty cycle
di=0.5;                                          % initial value
df=0.7;                                          % final value
tstep=0.02;                                      % time of step (s)

% load
Rload=10.0;                                      % load resistance (Ohms)
Yload=1/Rload;                                   % load conductance (Mhos)

% model parameters
Pdcdc.L=5e-3;                                    % inductor inductance (H)
Pdcdc.rL=0.5;                                    % inductor resistance (Ohms)
Pdcdc.C=1000e-6;                                 % capacitor capacitance (F)

% initial condition
vouti=vin/(Pdcdc.rL/(Rload*di)+di);              % initial output voltage (V)
ili=vouti/(di*Rload);                            % initial inductor current (A)
Pdcdc.xic=[ili vouti];                           % initial state vector
```

savm_dcdc


savm_dcdc
File Edit View Display Diagram Simulation Analysis Code Tools Help
0.1
Normal
Model Browser
savm_dcdc
savm_dcdc
Function Block Parameters: DC/DC Converter - AVM
(mask)
Parameters
(Double Click)
Parameter Structure
Pdcdc
Waveform Structure
DCDC_Waveforms
OK
Cancel
Help
Apply
vin
Input Voltage
Duty Cycle
vin
d
iout
vout
io
DC/DC Converter - AVM
Scope
Scope1
Yload
Load Admittance
R
Ready
100%
ode4