Numerical Algorithms for ODEs/DAEs (Transient Analysis)

Solving Differential Equation Systems

$$\frac{d}{dt}\vec{q}\,(\vec{x}(t)) + \vec{f}\,(\vec{x}(t)) + \vec{b}(t) = \vec{0}$$

- DAEs: many types of solutions useful
  - ✔ **DC steady state**: no time variations
  - **transient**: ckt. waveforms changing with time
  - **periodic steady state**: changes periodic w time
    - linear(ized): all sinusoidal waveforms: **AC analysis**
    - nonlinear steady state: **shooting**, **harmonic balance**
  - **noise analysis**: random/stochastic waveforms
  - **sensitivity analysis**: effects of changes in circuit parameters

Transient Analysis

$$\frac{d}{dt}\vec{q}(\vec{x}(t)) + \vec{f}(\vec{x}(t)) + \vec{b}(t) = \vec{0}$$

- What
  - inputs b(t) changing with time
    - find waveforms of x(t) as they change with time
- Why
  - most general analysis typically needed
  - sine wave, pulse, etc. inputs typical in many applications
- How
  - solve DAE using numerical methods
    - “discretize time”: replace d/dt term
    - convert DAE to nonlinear algebraic equation at each discrete time point
    - solve this using NR

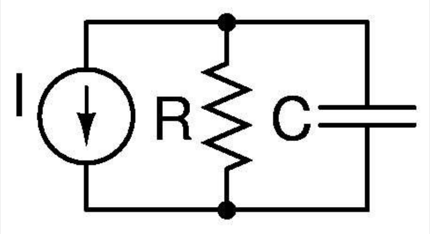
Solving DAEs: Preliminaries

- Given a DAE: does it have a solution?
  - Depends. Various conditions need to hold.
- Easier to analyze if DAEs are really ODEs
  - Ordinary Differential Equations: i.e., $\vec{q}(\vec{x}) \equiv \vec{x}$

$$\frac{d}{dt}\vec{x}(t) + \vec{f}(\vec{x}(t)) + \vec{b}(t) = \vec{0} \Rightarrow \quad \frac{d}{dt}\vec{x}(t) = \vec{g}(\vec{x}, t)$$

  - Existence/Uniqueness conditions well known for ODEs
    - f(x) needs to be Lipschitz
    - device models must be smooth / bounded / “physically reasonable”
      - watch those if conditions, 1/x, log(x), sqrt(x), etc. terms!
- DAEs? Much more involved
  - in practice: modifications of ODE methods + heuristics

Analytical Exemplar (Model Problem)



$$A \underset{k_2}{\overset{k_1}{\rightleftharpoons}} B$$

$$C\frac{de}{dt} = -\frac{e}{R} - I(t)$$

$$\frac{d[A]}{dt} = -(k_1 + k_2)[A] + k_2$$

$$\dot{x} = \lambda x + b(t)$$

- Useful because
  - has analytical solution
  - vector linear systems reducible to this form
  - locally approximates nonlinear systems
- Prototype for
  - stability analysis of ODE solution methods

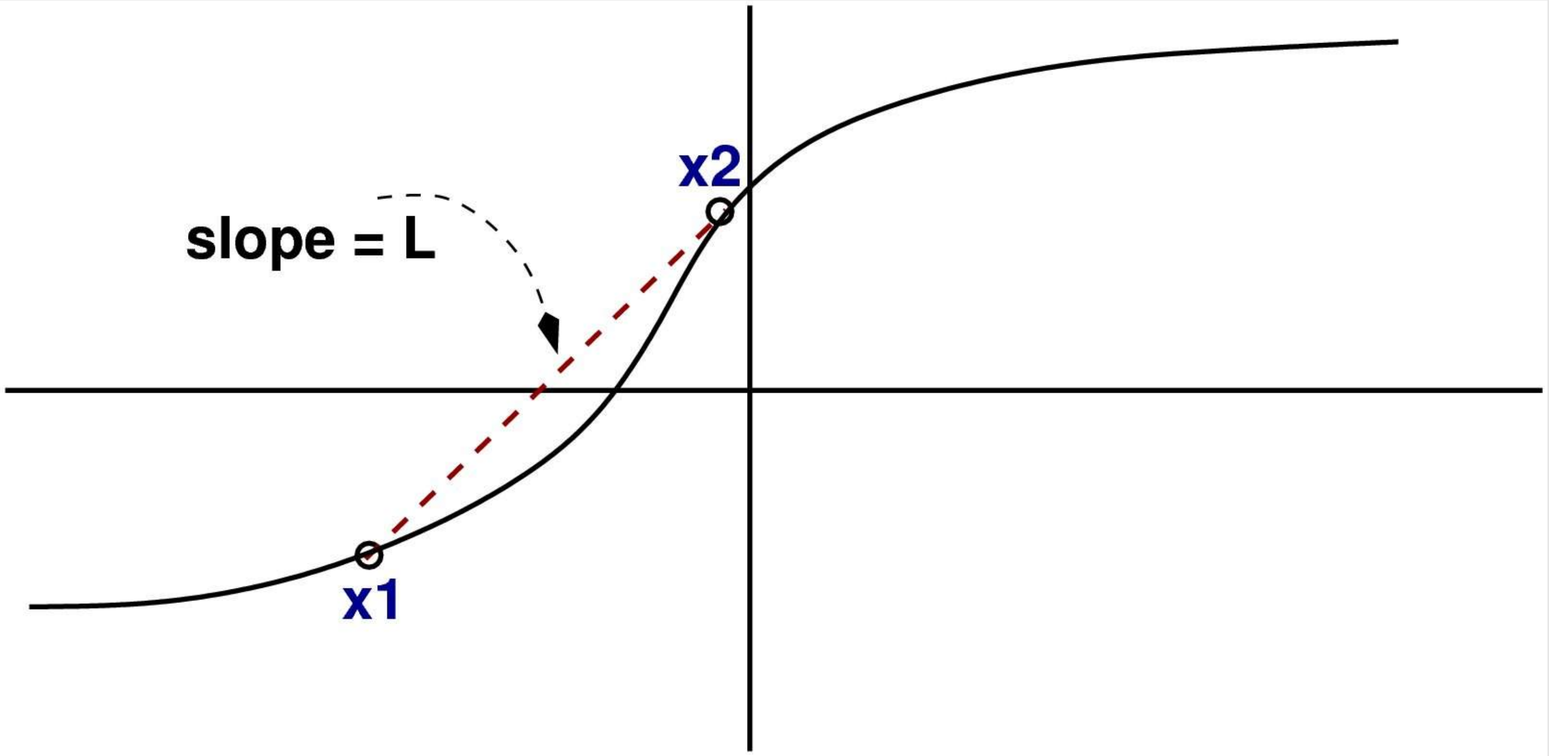
Existence and Uniqueness

- Does a solution exist? Examples:
  - $\dot{x} = \lambda x + b(t)$
    - yes: $x(t) = x(t_0)e^{\lambda(t-t_0)} + \int_0^t e^{\lambda(t-\tau)} b(\tau)\, d\tau$
    - Solution is unique, given an initial condition
  - $\dot{x} = -\frac{1}{2x}, \quad x(0) = x_0$
    - $x(t) = \sqrt{x_0^2 - t}$: no solution for $t > x_0^2$
  - $\dot{x} = \frac{3}{2} x^{\frac{1}{3}}$
    - $x(t) = \begin{cases} 0, & 0 <= t <= k \\ (t-k)^{\frac{3}{2}}, & t > k \end{cases}$, for ANY $k > 0$!
    - Infinite number of solutions!

Existence/Uniqueness Theorem

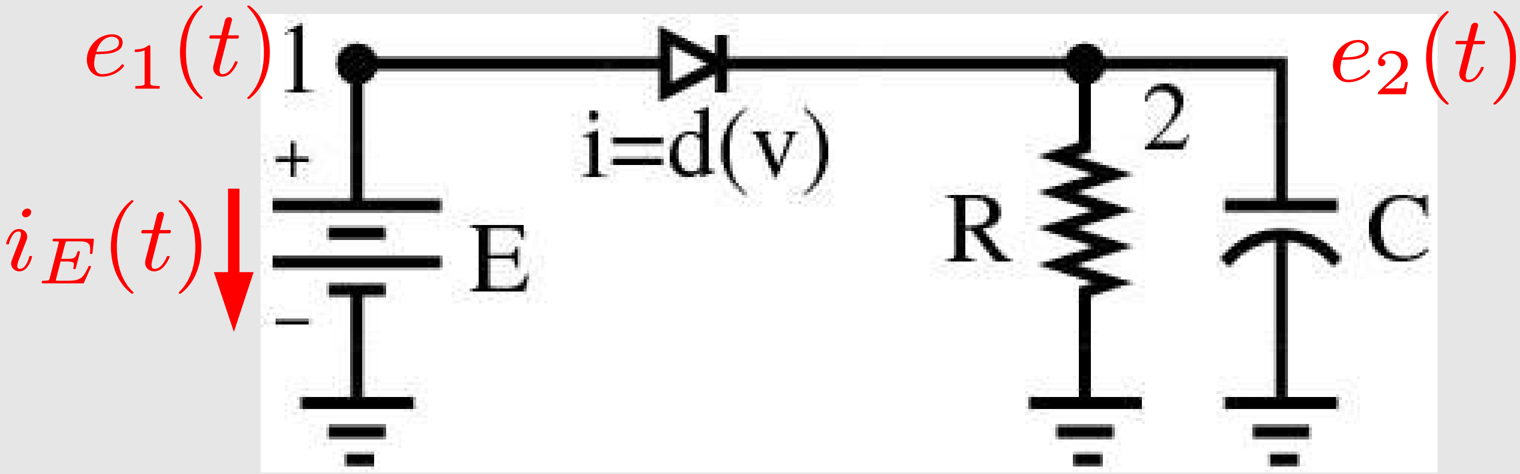
- Picard-Lindelöf Theorem (roughly)
  - If $\vec{g}\left(\vec{x}, t\right)$ is
    - **defined over all t, x**
    - **Lipschitz for all x**
  - then $\dot{\vec{x}}(t) = \vec{g}\left(\vec{x}, t\right), \quad \vec{x}(t_0) = \vec{x}_0$ has a **unique global solution**
- $\vec{g}\left(\vec{x}, t\right)$ **is Lipschitz if**
  - there exists some <u>finite</u> L such that:
    - $\|\vec{g}\left(\vec{x}_1, t\right) - \vec{g}\left(\vec{x}_2, t\right)\| < L\|\vec{x}_1 - \vec{x}_2\|, \quad \forall \vec{x}_1, \vec{x}_2$

The Lipschitz Condition



- Linear systems: Lipschitz
- $\sqrt[3]{x}$, $\dfrac{1}{x - x_0}$: not Lipschitz

Is this globally Lipschitz?



n1 KCL: $i_E + d(e_1 - e_2) = 0$

n2 KCL: $-d(e_1 - e_2) + \frac{e_2}{R} + \frac{d}{dt}(Ce_2) = 0$

E BCR: $e_1 - E = 0$

n2 KCL: $\frac{d}{dt}e_2 + \frac{e_2}{RC} - \frac{d(E(t) - e_2)}{C} = 0$

Solving ODEs: Overview of Strategy

- Discretize the time axis (starting from, e.g., t=0)
  - $t_0 = 0, t_1, t_2, \cdots, t_N$
- Approximate d/dt term by finite difference
  - e.g., $\frac{d\vec{x}}{dt} \simeq \frac{\vec{x}_i - \vec{x}_{i-1}}{t_i - t_{i-1}}$
- ODE becomes algebraic nonlinear equation

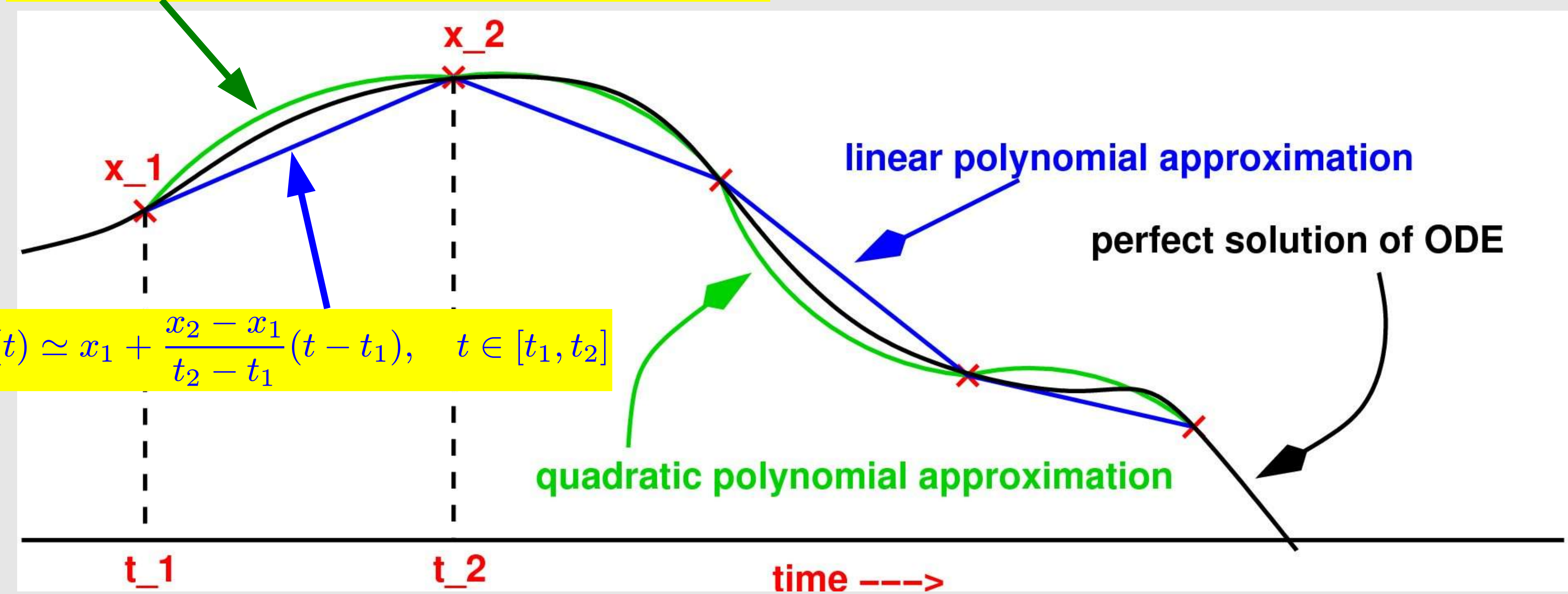
$$\underbrace{\frac{\vec{x}_i - \vec{x}_{i-1}}{t_i - t_{i-1}} + \vec{f}(\vec{x}_i) + \vec{b}(t_i)}_{\vec{h}_i(\vec{x}_i)} = \vec{0}$$

- Solve using Newton-Raphson
  - repeat for i=1,2,3,...,N
  - using previous solution (or initial condition) at t_{i-1}
- Key issues:
  - how to discretize? errors/accuracy? computation? coding?

Piecewise Polynomial Approximation

- Using locally polynomial bases
  - assume: ODE solution is locally polynomial
  - characterize polynomial with a **few numbers**
    - ➔ e.g., samples at different points
  - find those numbers so that the **local polynomial satisfies the ODE**

$$x(t) \simeq x_1 + c_1(t - t_1) + c_2(t - t_1)^2, \quad t \in [t_1, t_2]$$



$$x(t) \simeq x_1 + \frac{x_2 - x_1}{t_2 - t_1}(t - t_1), \quad t \in [t_1, t_2]$$

Linear Polynomials: FE and BE

- Use of **locally linear** approximations for x(t)
  - $\vec{x}(t) \simeq x_1 + \frac{\vec{x}_2 - \vec{x}_1}{t_2 - t_1}(t - t_1), \quad t \in [t_1, t_2]$
  - Knowns: t_1, t_2, x_1
    - → (x_1 known from, e.g., initial condition)
  - Unknown: $\mathbf{x_2}$
  - Use linear approx. to express $\dot{\vec{x}}(t)$: $\dot{\vec{x}}(t) \simeq \frac{\vec{x}_2 - \vec{x}_1}{t_2 - t_1}$
  - Enforce ODE $\dot{\vec{x}}(t) = \vec{g}(\vec{x}, t)$ at t_1: **Forward Euler (FE)**
    - → $\dot{\vec{x}}(t_1) = \vec{g}(\vec{x}_1, t_1)$: $\boxed{\frac{\vec{x}_2 - \vec{x}_1}{t_2 - t_1} = \vec{g}(\vec{x}_1, t_1)}$
  - Enforce ODE $\dot{\vec{x}}(t) = \vec{g}(\vec{x}, t)$ at t_2: **Backward Euler (BE)**
    - → $\dot{\vec{x}}(t_1) = \vec{g}(\vec{x}_2, t_2)$: $\boxed{\frac{\vec{x}_2 - \vec{x}_1}{t_2 - t_1} = \vec{g}(\vec{x}_2, t_2)}$

The Forward Euler Method

$$\dot{\vec{x}}(t_{i-1}) = \vec{g}\left(\vec{x}_{i-1}, t_{i-1}\right): \frac{\vec{x}_i - \vec{x}_{i-1}}{t_i - t_{i-1}} = \vec{g}\left(\vec{x}_{i-1}, t_{i-1}\right)$$

$$\Rightarrow \boxed{\vec{x}_i = \vec{x}_{i-1} + (t_i - t_{i-1})\vec{g}\left(\vec{x}_{i-1}, t_{i-1}\right)}$$

- **Explicit** integration method
  - x_2 available explicitly in terms of knowns
    - **nonlinear solve not needed** (single eval of g(.,.) suffices)
    - (we'll see later) **not numerically stable**
    - (we'll see later) **does not work for DAEs**
- **"Time stepping" ODE solution process**

1) start with initial condition: $x_0 = x(t_0)$, i=1

2) find x_i using FE equation, above

3) increment i; stop if t_i>stoptime, else goto 2

The Backward Euler Method

$$\dot{\vec{x}}(t_i) = \vec{g}\left(\vec{x}_i, t_i\right) : \frac{\vec{x}_i - \vec{x}_{i-1}}{t_i - t_{i-1}} = \vec{g}\left(\vec{x}_i, t_i\right) \Rightarrow \boxed{\frac{\vec{x}_i - \vec{x}_{i-1}}{t_i - t_{i-1}} - \vec{g}\left(\vec{x}_i, t_i\right) = \vec{0}}$$

- **Implicit** integration method
  - **Finding x_2 requires nonlinear solution (N-R)**
    - (we'll see later) **numerically (over)stable**
    - (we'll see later) **does work for DAEs**
- **"Time stepping" ODE solution process**

1) start with initial condition: $x_0 = x(t_0)$, i=1

2) find x_i using BE equation, above

   - solve it using Newton-Raphson

3) increment i; stop if t_i>stoptime, else goto 2

The Trapezoidal Method

- "Average" FE and BE:
  - $$\begin{array}{rl} & \dot{\vec{x}}(t_1) = \vec{g}\left(\vec{x}_1, t_1\right) \\ + & \dot{\vec{x}}(t_2) = \vec{g}\left(\vec{x}_2, t_2\right) \\ \hline \end{array}$$
    $$\frac{\dot{\vec{x}}(t_1) + \dot{\vec{x}}(t_2)}{2} = \frac{\vec{g}\left(\vec{x}_1, t_1\right) + \vec{g}\left(\vec{x}_2, t_2\right)}{2}$$
- PWL assumption $\Rightarrow \dot{\vec{x}}(t_1) = \dot{\vec{x}}(t_2) = \dfrac{\vec{x}_2 - \vec{x}_1}{t_2 - t_1}$
- **Trapezoidal Method**: $\dfrac{\vec{x}_2 - \vec{x}_1}{t_2 - t_1} = \dfrac{\vec{g}\left(\vec{x}_1, t_1\right) + \vec{g}\left(\vec{x}_2, t_2\right)}{2}$
  - at i^{th} timestep:

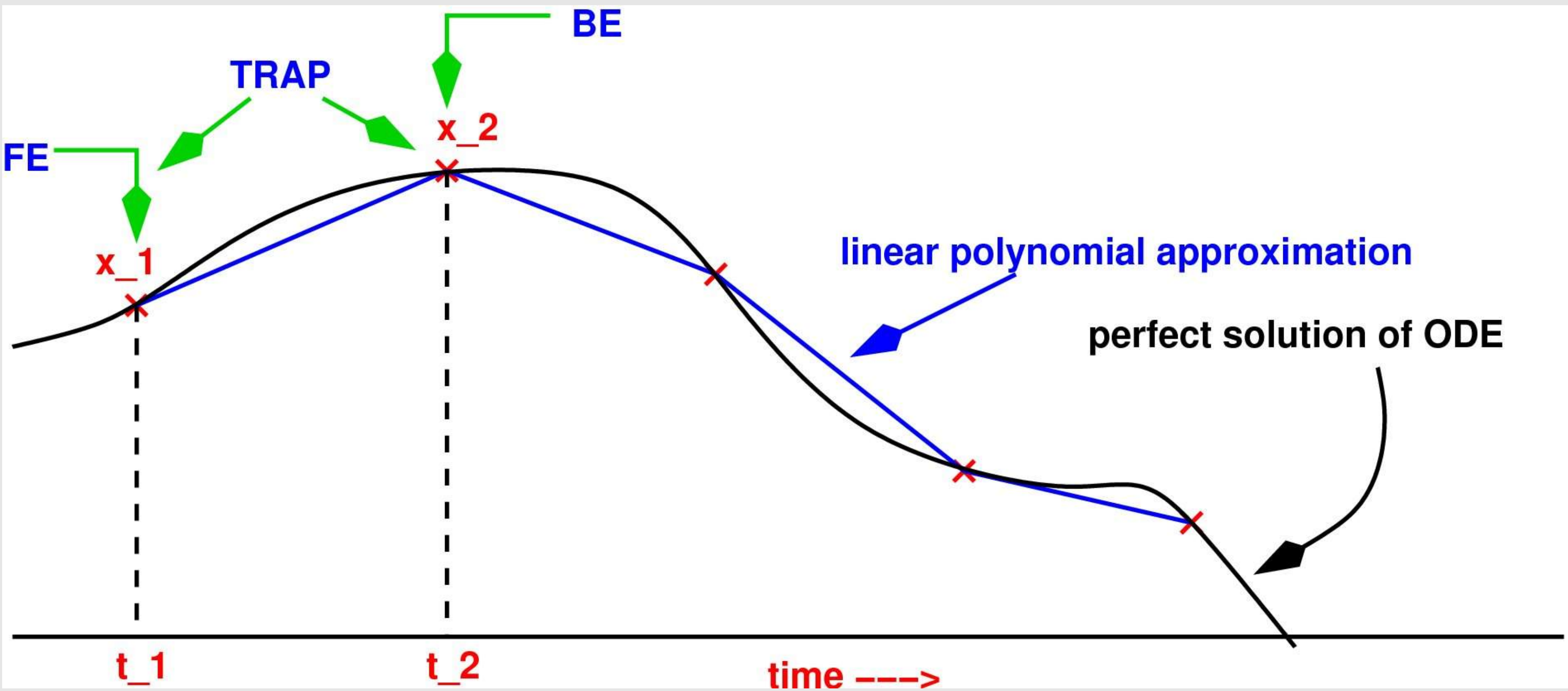
$$\frac{\vec{x}_i - \vec{x}_{i-1}}{t_i - t_{i-1}} = \frac{\vec{g}\left(\vec{x}_{i-1}, t_{i-1}\right) + \vec{g}\left(\vec{x}_i, t_i\right)}{2}$$

Implicit (N-R needed)

Perfectly stable numerically
(we'll see later)

FE, BE, TRAP: Pictorial Summary

- Difference: where/how ODE is “enforced”



Linear Multi Step (LMS) Methods

- FE: $\vec{x}_i - \vec{x}_{i-1} = h_i\, \vec{g}\left(\vec{x}_{i-1}, t_{i-1}\right), \quad h_i \equiv t_i - t_{i-1}.$
- BE: $\vec{x}_i - \vec{x}_{i-1} = h_i\, \vec{g}\left(\vec{x}_i, t_i\right)$
- TRAP: $\vec{x}_i - \vec{x}_{i-1} = h_i\, \dfrac{\vec{g}\left(\vec{x}_i, t_i\right) + \vec{g}\left(\vec{x}_{i-1}, t_{i-1}\right)}{2}$
- Generic: $\alpha_0 \vec{x}_i + \alpha_1 \vec{x}_{i-1} = \beta_0 \vec{g}\left(\vec{x}_i, t_i\right) + \beta_1 \vec{g}\left(\vec{x}_{i-1}, t_{i-1}\right)$
  - FE: $\alpha_0 = 1, \alpha_1 = -1, \beta_0 = 0, \beta_1 = h_i$
  - BE: $\alpha_0 = 1, \alpha_1 = -1, \beta_0 = h_i, \beta_1 = 0$
  - TRAP: $\alpha_0 = 1, \alpha_1 = -1, \beta_0 = \dfrac{h_i}{2}, \beta_1 = \dfrac{h_i}{2}$
- *p*th-order Linear Multi Step integration formula

$$\sum_{i=0}^{p} \alpha_i\, \vec{x}_{n-i} = \sum_{i=0}^{p} \beta_i\, \vec{g}\left(\vec{x}_{n-i}, t_{n-i}\right)$$

Use of ODE LMS Methods for DAEs

$$\frac{d}{dt}\vec{q}(\vec{x}(t)) + \vec{f}(\vec{x}(t)) + \vec{b}(t) = \vec{0} \Rightarrow \frac{d}{dt}\vec{q}(\vec{x}(t)) = \vec{g}(\vec{x}, t)$$

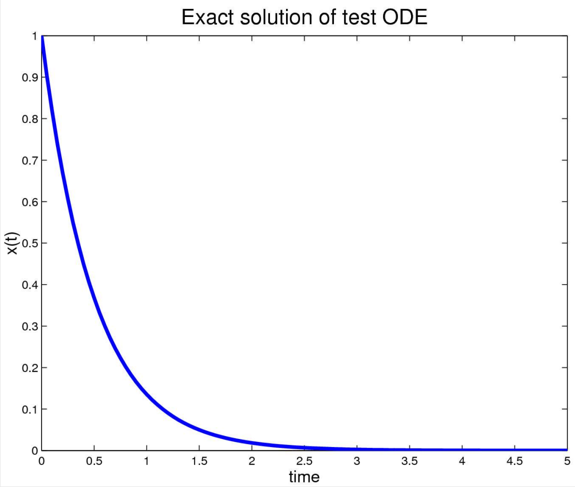
- The (simple) idea: re-do all derivations with $\vec{q}(t) \equiv \vec{q}(\vec{x}(t))$ on LHS, instead of $\vec{x}(t)$.
  - Generic LMS for p=1 (1-step LMS):
    - $\alpha_0\vec{q}(\vec{x}_i) + \alpha_1\vec{q}(\vec{x}_{i-1}) = \beta_0\vec{g}(\vec{x}_i, t_i) + \beta_1\vec{g}(\vec{x}_{i-1}, t_{i-1})$
  - BE: $\vec{q}(\vec{x}_i) - \vec{q}(\vec{x}_{i-1}) = h_i\ \vec{g}(\vec{x}_i, t_i)$
  - TRAP: $\vec{q}(\vec{x}_i) - \vec{q}(\vec{x}_{i-1}) = h_i\ \dfrac{\vec{g}(\vec{x}_i, t_i) + \vec{g}(\vec{x}_{i-1}, t_{i-1})}{2}$
  - FE: $\vec{q}(\vec{x}_i) - \vec{q}(\vec{x}_{i-1}) = h_i\ \vec{g}(\vec{x}_{i-1}, t_{i-1})\,.$

Implicit (N-R needed)

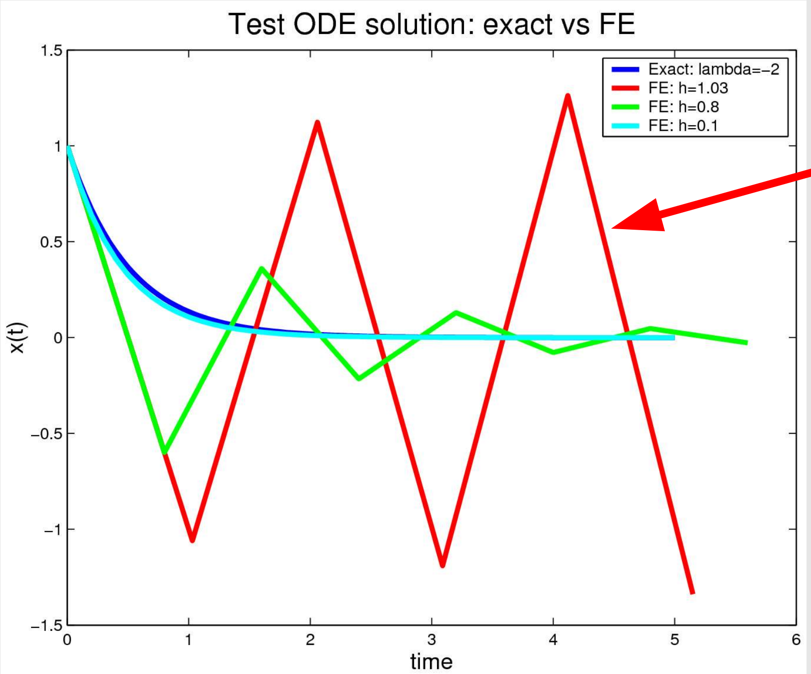
Implicit?

Stability of LMS Methods

- Test ODE: $\dot{x}(t) = \lambda x(t), \quad x(0) = x_0.$
  - exact solution: $x(t) = x_0 e^{\lambda t}.$
    - solution decays if $\lambda < 0.$



- What do FE/BE/TRAP produce?



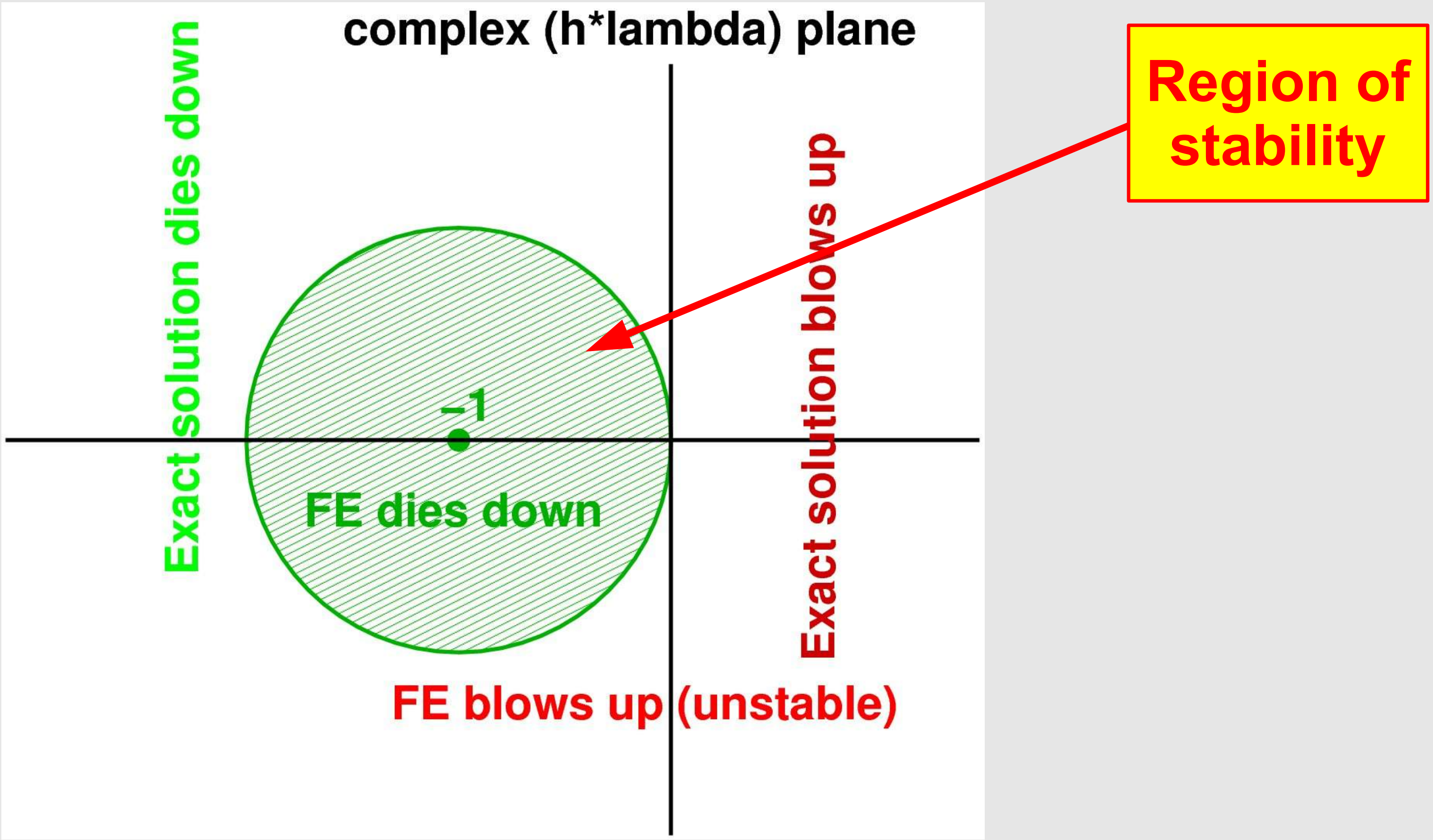
FE is qualitatitively wrong

Why FE Explodes

- FE (with constant timestep h)
  - $x_n = (1 + h\lambda)x_{n-1} \Rightarrow \boxed{x_n = x_0(1 + h\lambda)^n}.$
  - if $|1 + h\lambda| > 1$, solution blows up w.r.t n.
    - solution is **qualitatively wrong** for $\lambda < 0.$
- Basic requirement for FE
  - h should be small enough s.t. $|1 + h\lambda| < 1.$
  - Example: $\lambda = 10^9 \Rightarrow \boxed{h < 2 \times 10^{-9}}.$
    - **FE limited to small timesteps**
      - for even "qualitatitive" accuracy
- FE said to be (numerically) **unstable**
  - if $|1 + h\lambda| > 1$

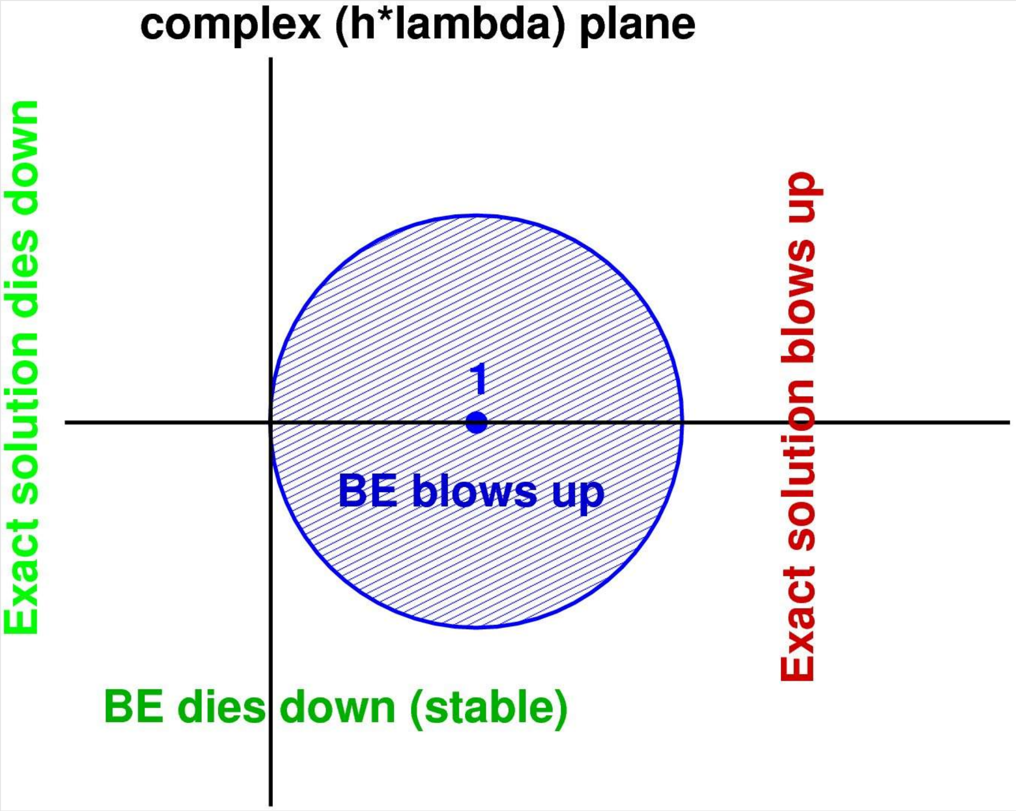
FE: Stability Picture for Complex λ

- In general: eigenvalues can be complex
  - stability condition $|1 + h\lambda| < 1$: **circle** in hλ plane



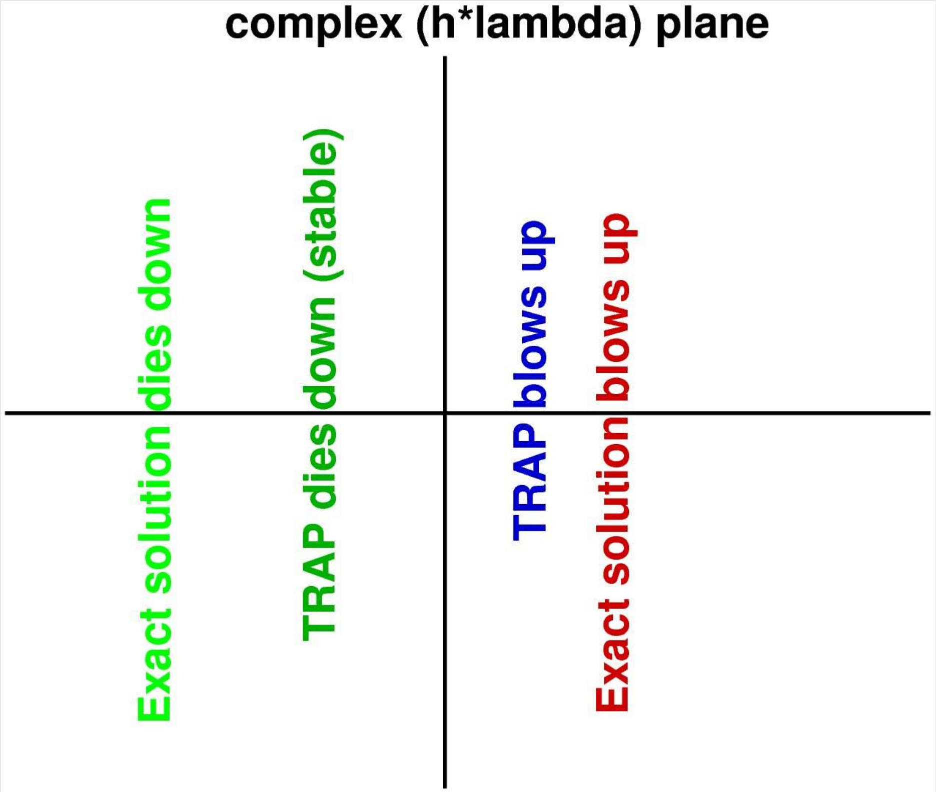
Stability of BE

- BE with constant timestep h
  - $x_n(1 - h\lambda) = x_{n-1} \Rightarrow \boxed{x_n = x_0 \frac{1}{(1 - h\lambda)^n}}.$
  - solution will die down if $|1 - h\lambda| > 1.$
    - i.e., all of left half plane.
  - but also
    - much of right half plane
    - **BE is overstable**
      - **OK only within circle**
- **Applications**
  - **λ<0 more important**
  - **much better than FE**



Stability of TRAP

- TRAP with constant timestep h
  - $x_n\left(1-\frac{h\lambda}{2}\right) = x_{n-1}\left(1+\frac{h\lambda}{2}\right) \Rightarrow \boxed{x_n = x_0\left(\frac{2+h\lambda}{2-h\lambda}\right)^n}.$
  - solution will die down if Re(λ) < 0
  - solution will blow up if Re(λ) > 0
    - ➔ i.e, **perfect stability**
- Also very accurate
  - (we'll see later)
- Only concern
  - DAE initial consistency
    - ➔ (we'll see later)
    - ➔ (easy practical workaround)



Accuracy and Truncation Error

- Even if stable: is the method accurate?
- Test problem exact solution: $x(nh) = x_0 e^{\lambda n h}$.
- How do numerical method solutions compare?
  - FE: $x(nh) \simeq x_0 (1 + h\lambda)^n$.
  - BE: $x(nh) \simeq x_0 \dfrac{1}{(1 - h\lambda)^n}$.
  - TRAP: $x(nh) \simeq x_0 \left(\dfrac{2 + h\lambda}{2 - h\lambda} \right)^n$.
- None are identical, but how different?

Taylor Expansion Error

- Exact: $x(nh) = x_0 e^{\lambda n h} = x_0 \left(1 + nh\lambda + \frac{n^2 h^2 \lambda^2}{2!} + \cdots\right)$
  - FE: $x(nh) \simeq x_0 \left(1 + nh\lambda + \frac{n(n-1)}{2} h^2 \lambda^2 + \cdots\right)$
    - first-order accurate
  - BE: $x(nh) \simeq x_0 \left(1 + nh\lambda + \frac{n(n+1)}{2} h^2 \lambda^2 + \cdots\right)$
    - also first-order accurate
  - TRAP: $x(nh) \simeq x_0 \left(1 + nh\lambda + \frac{n^2}{2} h^2 \lambda^2 + \cdots\right)$
    - **second-order accurate**

Transient: Timestep Control

- Choosing the next timestep dynamically
- LTE based control
  - apply LTE formulae to estimate error
    - change timestep to meet some specified error
  - error specification: like reltol-abstol (reltol=percentage)
    - make sure these are looser than NR tolerances!
  - element-by-element vs vector norm based
    - 2-norm vs max norm; DAE issues
  - (change integration method based on timestep)
- NR convergence based control
  - cut timestep if NR does not converge
    - also: increase maxiter
  - increase timestep if NR converges "too easily"
    - also: decrease maxiter

ODE/DAE Packages Out There

- MATLAB has various ODE/DAE integrators
  - ode23, ode45, ...; ode23t, ode15s, ...
- DASSL/DASPK: general purpose DAE packages
  - Linda Petzold, UCSB
  - Fortran
  - some tweaking helpful for circuit applications
- Easy (and often worthwhile) to roll your own
  - tweaking, special heuristics, debugging, ...

Transient: Other Important Issues

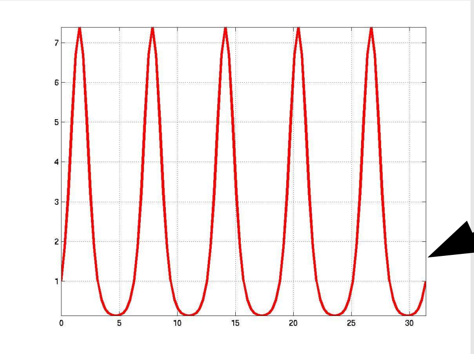
- What integration method to use?
  - stability?
  - nonlinear solution required? (implicit vs explicit)
  - accuracy loss due to discretization?
    - Local Truncation Error (LTE)
    - higher order methods (more than 1 previous timepoint)
- Vast body of work on ODE integration
  - linear multi-step methods (LMS), Runge-Kutta, symmetric, symplectic, “energy-conserving”, etc.
- Stiff differential equations
  - different variables have very different time constants
    - stiffly stable methods allow you to take larger time steps
- DAE issues
  - initial condition consistency; stability; index; …
- Dynamic timestep control, NR heuristics, ...

Solving the Circuit's Equations

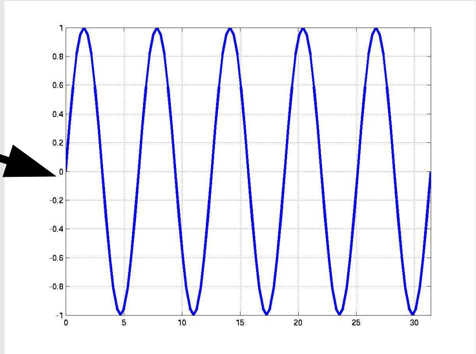
$$\frac{d}{dt}\vec{q}(\vec{x}(t)) + \vec{f}(\vec{x}(t)) + \vec{b}(t) = \vec{0}$$

- Ckt. DAEs: many types of solutions useful
  - ✔ **DC steady state**: no time variations
  - ✔ **transient**: ckt. waveforms changing with time
  - **periodic steady state**: changes periodic w time
    - linear(ized): all sinusoidal waveforms: **AC analysis**
    - nonlinear steady state: **shooting**, **harmonic balance**
  - **noise analysis**: random/stochastic waveforms
  - **sensitivity analysis**: effects of changes in circuit parameters

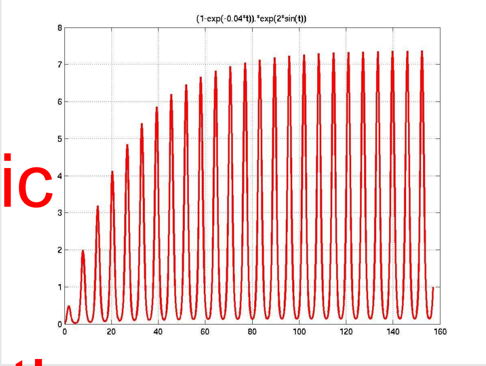
The Periodic Steady State Problem



$$\frac{d}{dt}\vec{q}(\vec{x}(t)) + \vec{f}(\vec{x}(t)) + \vec{b}(t) = \vec{0}$$



- What
  - inputs b(t) are periodic – e.g., sinusoidal
  - suppose “outputs” x(t) also become periodic
    - (happens for “stable” circuits and systems ...
    - … eventually – can take a long time)



  - want to find this <u>periodic steady state</u> directly
    - without using general/expensive transient analysis
- Why
  - audio amps, RF amps, mixers, oscillators, clocks, ...
    - ckt nonlinear => sinusoids will in general be distorted
  - linear circuits: frequency-domain analysis important
    - much easier and more insightful than transient
- How (for linear circuits): AC analysis