Simulating a Dynamic Model: Reviewing Numerical Integration

Aerospace Autonomy

References; Hamming, R. W. (1962).
Numerical methods for Scientists and Engineers

AERSP 497/597
Dr Simon W Miller
simon.w.miller@psu.edu

Basic premise:
Can propagate forward dynamics throughout time

- Common formulation using the system model

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$$

- Numerical integration finds

$$\boldsymbol{x}(t) = \boldsymbol{f}\big(\dot{\boldsymbol{x}}(t - \Delta t)\big)$$

Concept of Numerical Integration

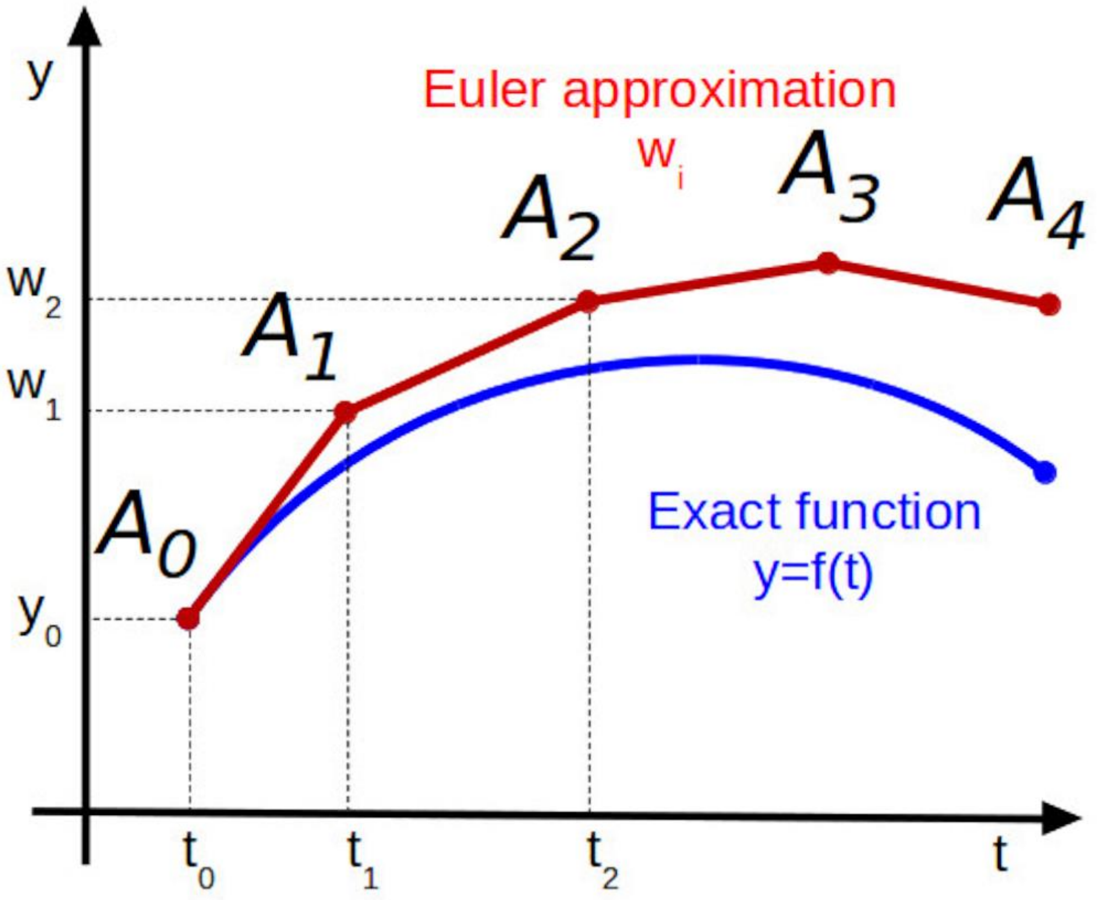
- Remember Taylor Series Expansion?

$$f(x + \Delta t) \approx f(x) + \Delta t\, f'(x) + \frac{\Delta t^2}{2!} f''(x) + \cdots$$

- First-Order Forward Euler (FOFE)

$$\boldsymbol{x}(t_{i+1}) = \boldsymbol{x}(t_i) + \dot{\boldsymbol{x}}(t_i) \cdot \Delta t + \mathcal{O}(\Delta t^2)$$

- The ‘classic’ vision of numerical integration
- But not the best algorithm…



Accuracy of Forward Euler

- Useful test: how well can it approximate

$$x(t) = e^{a\,t}$$
$$\dot{x}(t) = a\,e^{at} = a \cdot x(t)$$

- For Forward Euler:

$$\dot{x}(t) \approx x(t) + \Delta t\, x'(t) + \mathcal{O}(\Delta t^2)$$
$$x(t + \Delta t) \approx \underbrace{e^{at}} + \underbrace{\Delta t \cdot a e^{at}} = x(t)(1 + a\Delta t)$$

$$x(t + n\Delta t) = (1 + a\Delta t)^n \cdot\ x_{t=0}$$

- Compare the two, reality and approximation
  - And think about when Forward Euler will blow up…

Stability of Forward Euler

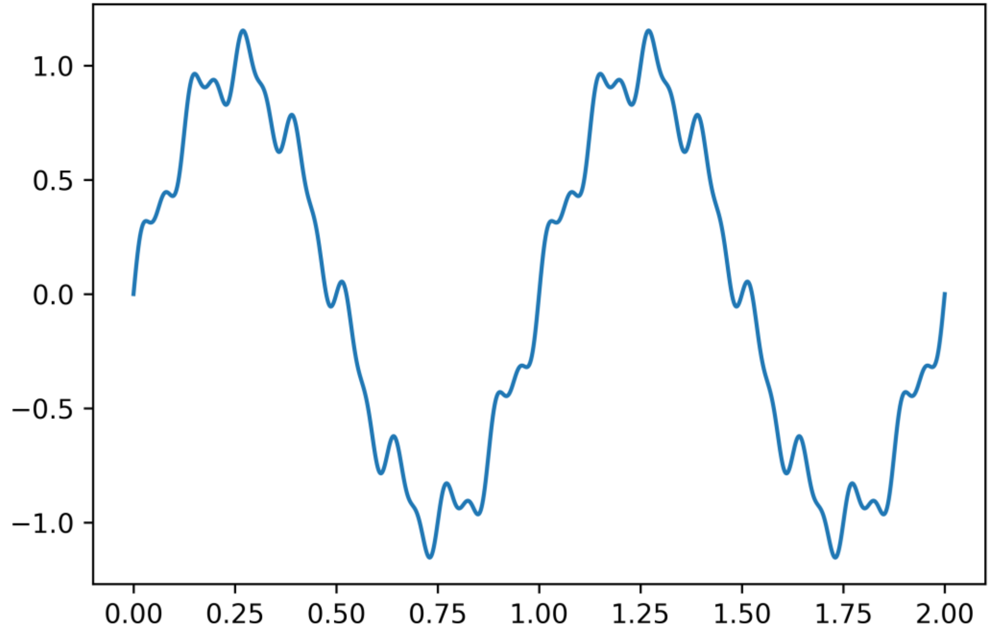
$$x(t) = e^{a\,t}$$

- If $a < 0$ (decay):
  - Solution can grow in magnitude if $1 + a\Delta t < -1$
  - Need small Δt if 'a' very negative

- If $a < 0$ (decay)
  - Solution will oscillate between positive & negative if
    $$-1 < 1 + a\Delta t < 0$$
  - Need small Δt if 'a' very negative

PennState

Here is the solution to the differential equation that we want to find numerically…



$$f'(x) = 2\pi\cos(2\pi x) + \frac{16\pi\cos(16\pi x)}{8} + \frac{32\pi\cos(32\pi x)}{16}$$

$$\therefore f(x) = \sin(2\pi x) + \frac{\sin(16\pi x)}{8} + \frac{\sin(32\pi x)}{16}$$

PennState

Let's look at just one small step ...

Analytic

$f_0(0) = 0$

$f_1(0 + 0.1) \approx f_0 + 0.1 \cdot f'(0)$

$= 0 + 1.88496$

$= 1.884956$

$f_2(0.1 + 0.1) \approx f_1 + 0.1 \cdot f'(0.1)$

$= 1.884956 + 0.194161$

$= 2.079117$

$f_3(0.2 + 0.1) \approx f_2 + 0.1 \cdot f'(0.2)$

$= 2.079117 + -0.119998$

$= 1.959119$

Forward-Euler

$f(0) = 0$

$f(0.1) = 0.432167$

$f(0.2) = 0.937024$

$f(0.3) = 0.965089$

Ummm...this is looking *bad!*

PennState

Let's look at just one small(er) step ...$\Delta t = 0.0125$

Analytic

$f_0(0) = 0$

$f_1(0 + \Delta t) \approx f_0 + \Delta t\, f'(0)$

$= 0.235619$

$f_2(\Delta t + \Delta t) \approx f_1 + \Delta t f'(\Delta t)$

$= 0.401727$

$f_3(2\Delta t + \Delta t) \approx f_2 + \Delta t f'(2\Delta t)$

$= 0.440030$

Forward-Euler

$f(0) = 0$

$f(\Delta t) = 0.211373$

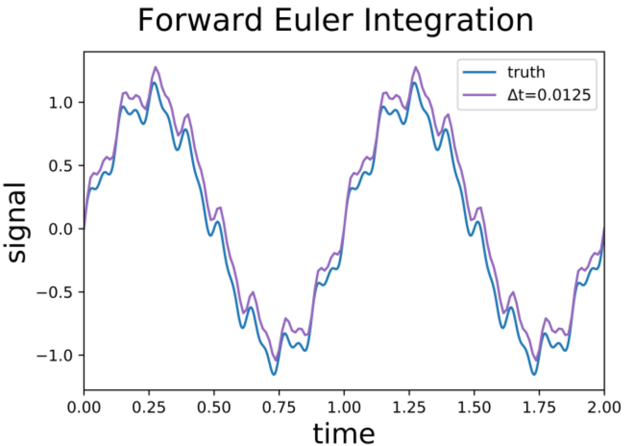
$f(2\Delta t) = 0.312053$

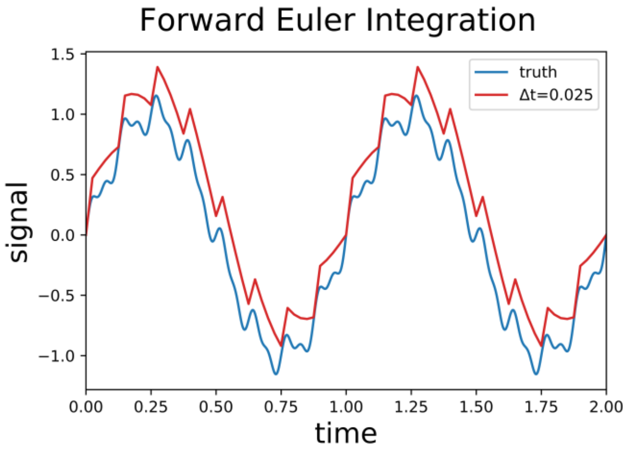
$f(3\Delta t) = 0.315591$

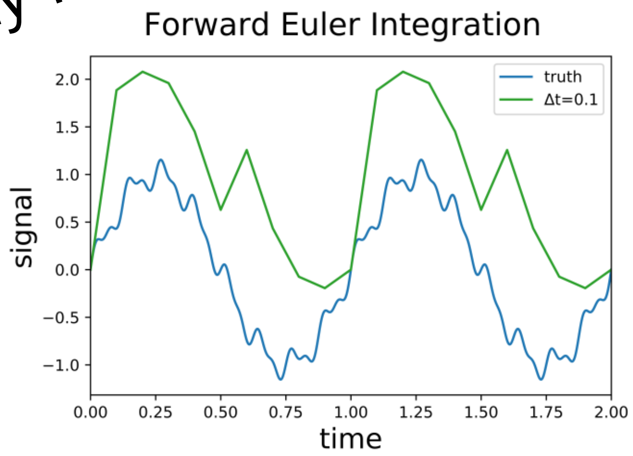
Well...that is *better*

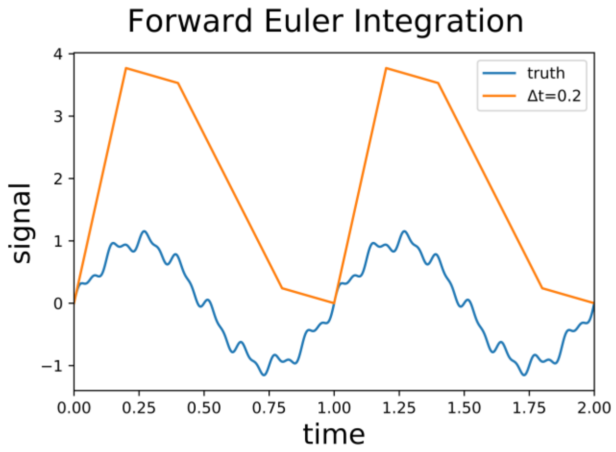
Applying Forward Euler integration … it loses stability as soon as it loses accuracy

← More better? Why not always?



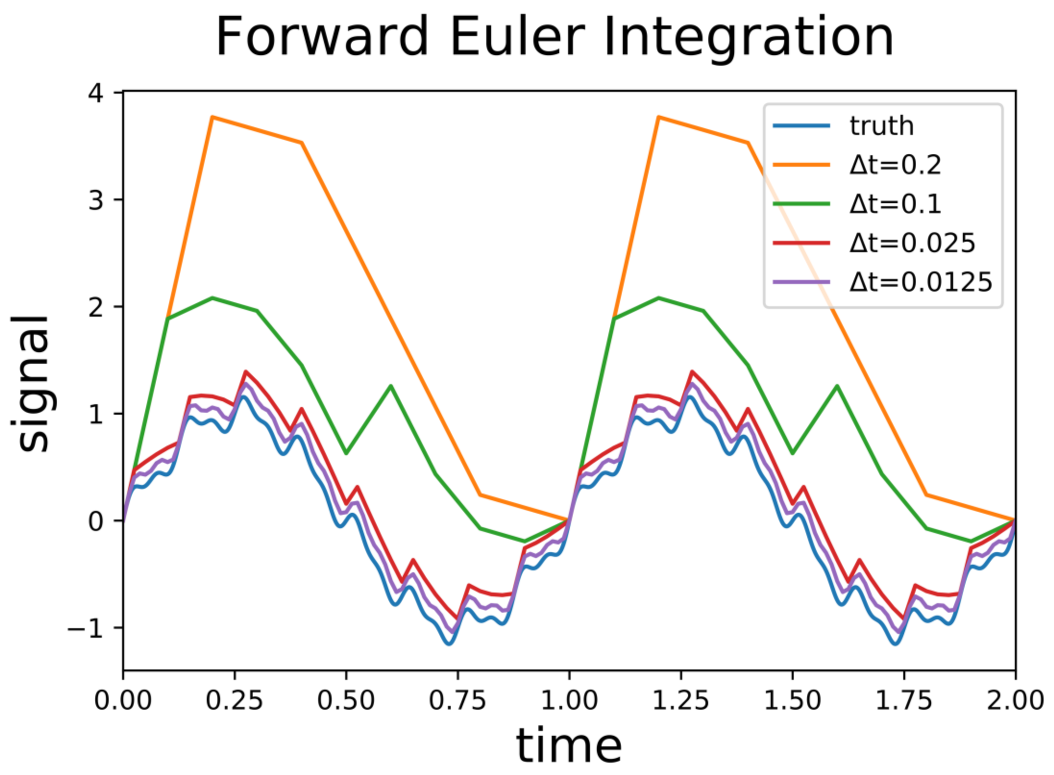






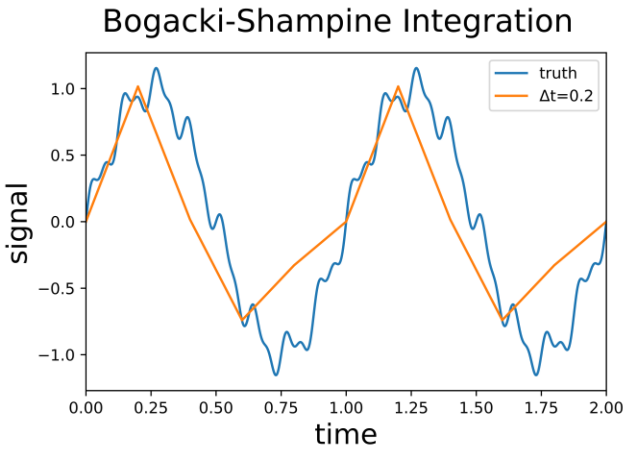
FOFE Integration with increasing timestep →

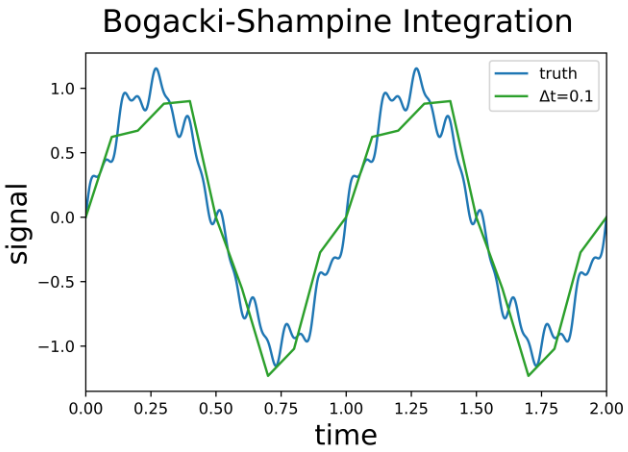
Laying them on top of the Truth

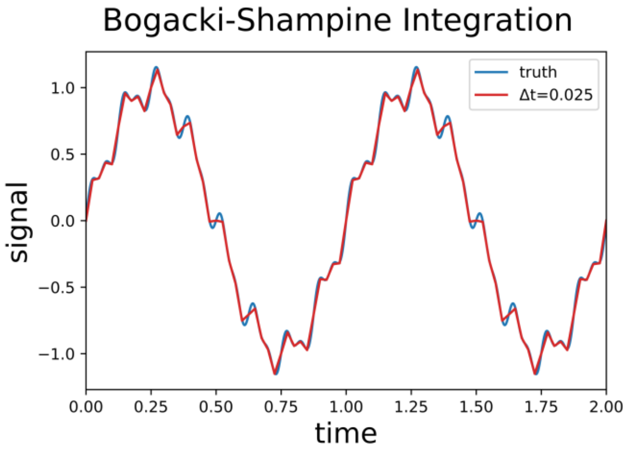


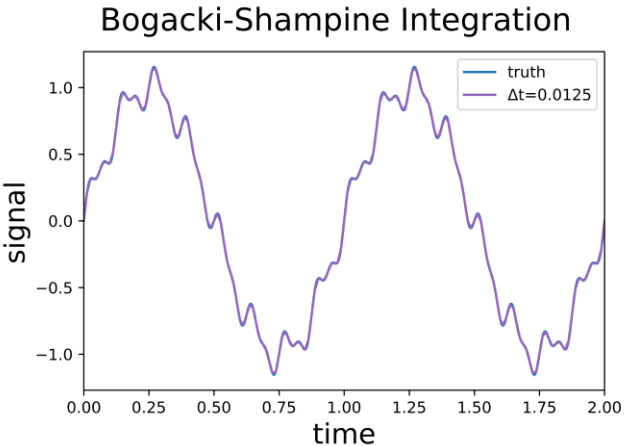
PennState

This type of integration maintains stability, even when inaccurate



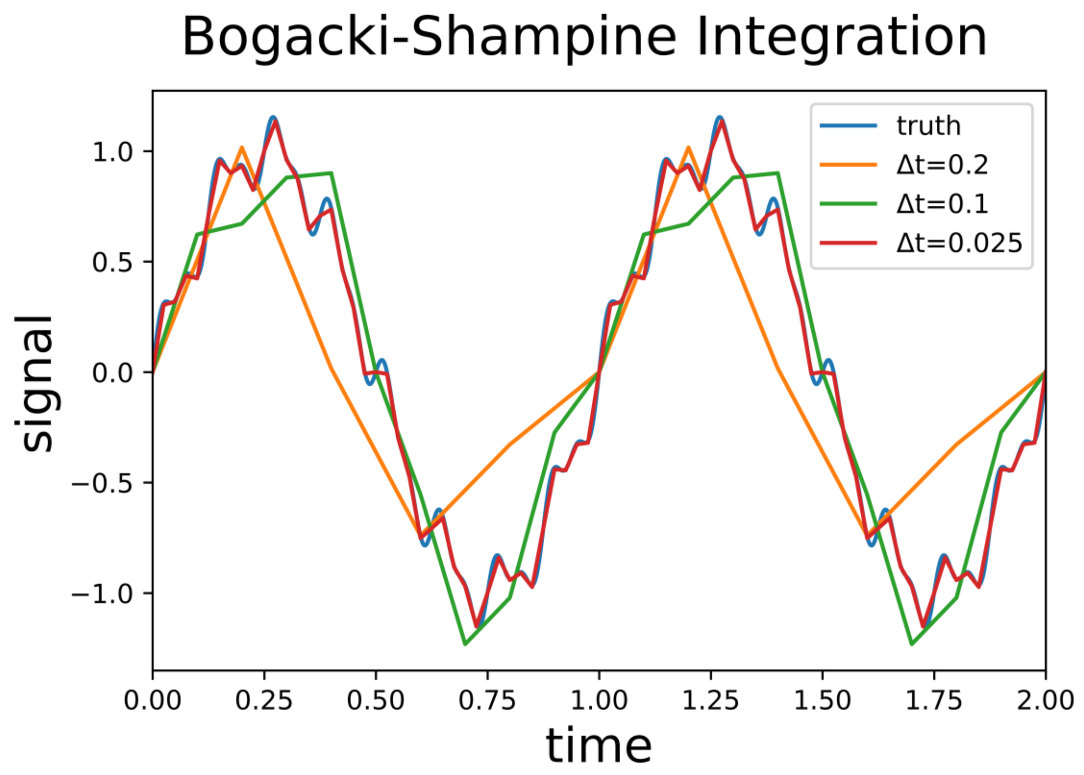






Shampine Integration with increasing timestep →

Laying them on top of the Truth



Numerical integration 'goodness' measured by

- Accuracy
  - The error $x_{n\Delta t} - x_n$ can be bounded by $\mathcal{O}(n\Delta t)^p$
  - Error can be reduced by:
    - Reducing Δt (almost always, except numerical stability!)
    - Using an algorithm with a higher order of accuracy p

- Stability
  - Do errors grow in relation to $x\{t\}$, or do they take on a life of their own?

- Simplicity, i.e. Flops
  - Computation time caused by calculating derivative

PennState

So, what's the point?

- There are many different 'integration methods'

- There are three performance metrics by which to evaluate them:
  - Accuracy, Stability, Simplicity

- Different integration methods score better-or-worse on these metrics
  - Some can score badly … on all of the metrics!

First-Order Backward Euler

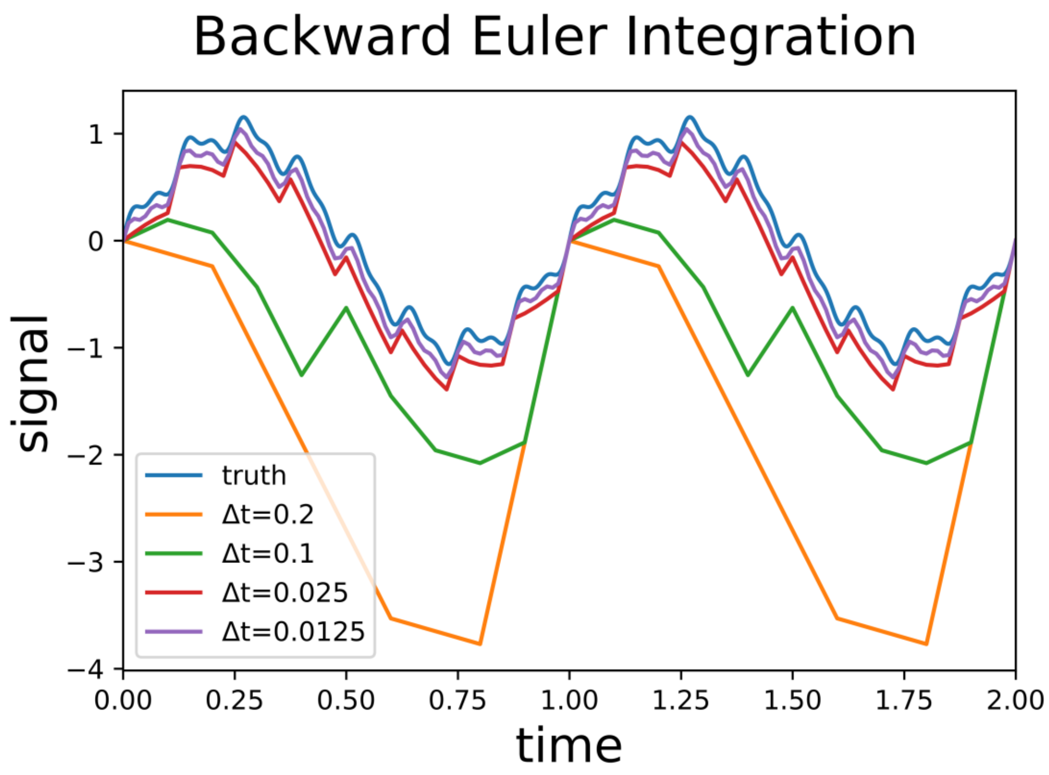
- Calculate derivative @ next time-step (instead of at current time)

$$x_{n+1} - \dot{x}\{x_{n+1}\}\Delta t = x_n, \qquad \rightarrow \quad x_{n+1} = \frac{1}{1 - a\Delta t} x_n$$

- Still Only First-Order Accuracy ($p = 1$)
  - But stable for all decay!
  - Stable for growth when $a\Delta t < 1$

PennState

Laying them on top of the Truth



Trapezoidal Rule

- Use contribution from derivative at this and next time-step

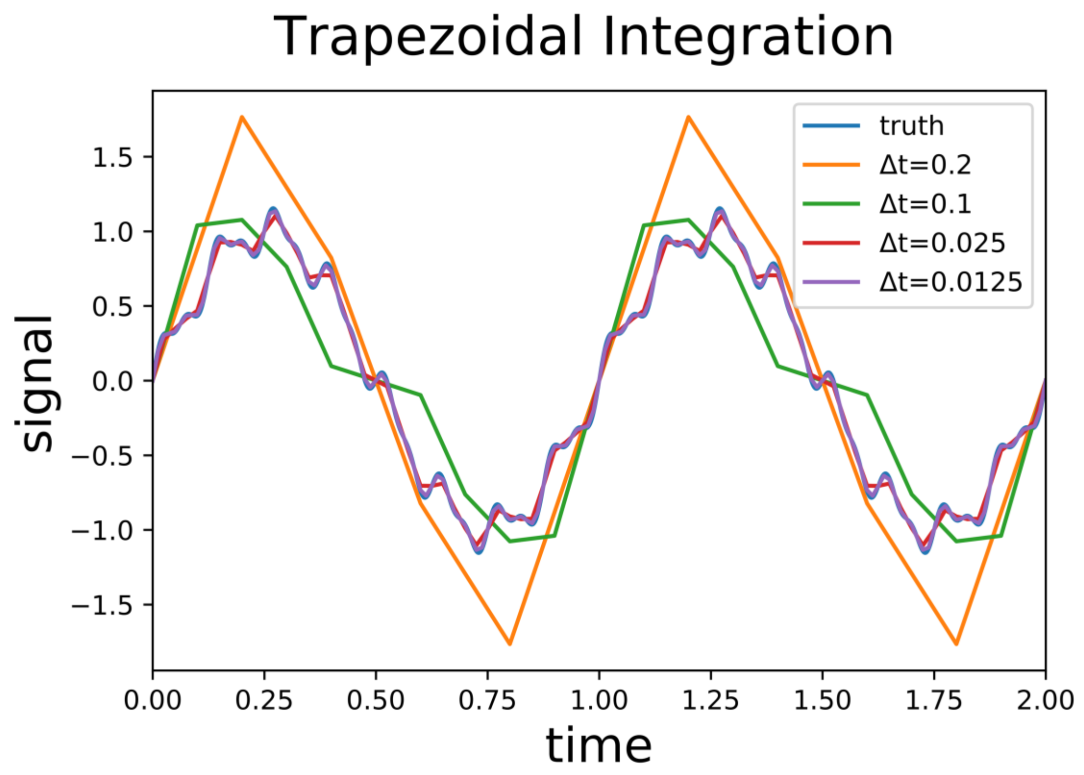
$$x_{n+1} - \frac{1}{2}\dot{x}\{x_{n+1}\}\Delta t = \frac{1}{2}\dot{x}\{x_n\}\Delta t + x_n$$
$$\rightarrow \quad x_{n+1} = x_n + \frac{1}{2}(\dot{x}\{x_{n+1}\} + \dot{x}\{x_n\})\Delta \mathrm{t}$$

- Second-Order Accuracy!

- Stability same as Backward Euler

PennState

This type of integration has *better* stability than FOFE



PennState

Problem with Methods to Date

- In reality, not looking at super simple functions like e^{at}
  - How to set Δt?
    - Predict 'fastest' derivative within system (greatest '$\boldsymbol{a}$')
    - Look for stability requirements (growth & decay)

- Error always grows to first/second order

- Backward Euler & Trapezoidal
  - More stable -- but how do you calculate the derivative of a *future state before you know what the future state is* in the first place? (Implicit routines)

PennState

Factors affecting numerical integration success

- Where in the time-step is the derivative calculated?

- How big is the time-step?

- More advanced routines…
  - Better forward estimate of derivative through consideration of multiple past derivatives
  - Better forward estimate of derivative throughout time-step
  - Better estimate of derivative through iteration
  - Adjustable time-step

Multi-Step Methods

- Estimate derivative from multiple past values

$$x_{n+1} = x_n + \left(c_n \dot{x}_n + c_{n-1} \dot{x}_{n-1} + \cdots + c_{n-p} \dot{x}_{n-p}\right) \Delta t$$

$$\sum_{n=1}^{p} c_i = 1$$

- Accuracy of order p
  - Creates accurate estimate of smoothly varying derivatives
  - Late response to sudden changes in derivative

PennState

Multi-Step Coefficients

- More coefficients for hyper-accuracy
  - e.g., Astronomers go Out to $p = 8$
- Adams–Bashforth methods related to MATLAB’s ode23 and ode45

	c_1	c_2	c_3	c_4	Stability Threshold	Error Constant
$\boldsymbol{P = 1}$	1				-2	1/2
$\boldsymbol{P = 2}$	3/2	-1/2			-1	5/12
$\boldsymbol{P = 3}$	23/12	-16/12	5/12		-6/11	3/8
$\boldsymbol{P = 4}$	55/24	-59/24	37/24	-9/24	-3/10	251/720

PennState

Predictor-Error-Corrector

- Forward Euler is EXPLICIT -- assumes can estimate forward, linearly, the derivative value
- Would be nice to compare derivative at end of time-step with derivative used to get there
- Suggests iterative process
  - P: use explicit multi-step process to predict x_{n+1}
  - E: use x_{n+1} to *evaluate* $\dot{x}_{n+1}$
  - C: extrapolate from x_n using $\dot{x}_{n+1}$ to *correct* x_{n+1} with implicit (backwards) method, e.g., Trapezoidal rule

Can repeat a set number of times, or until error is within bounds

PennState

Runge-Kutta Routines

- RK algorithms calculate derivative *multiple times* within the same time-step
- Unlike PEC methods, though, it does not iterate freely, i.e., we have a constrained iteration technique – constrained iterations

- RK2:

$$\dot{\boldsymbol{x}}_1 = \dot{\boldsymbol{x}}\{\boldsymbol{x}(t), t\}$$
$$\dot{\boldsymbol{x}}_2 = \dot{\boldsymbol{x}}\{\boldsymbol{x}(t) + \dot{\boldsymbol{x}}_1 \cdot \Delta t, t + \Delta t\}$$
$$\dot{\boldsymbol{x}}_{RK2} = \frac{1}{2}(\dot{\boldsymbol{x}}_1 + \dot{\boldsymbol{x}}_{\boldsymbol{2}})$$

PennState

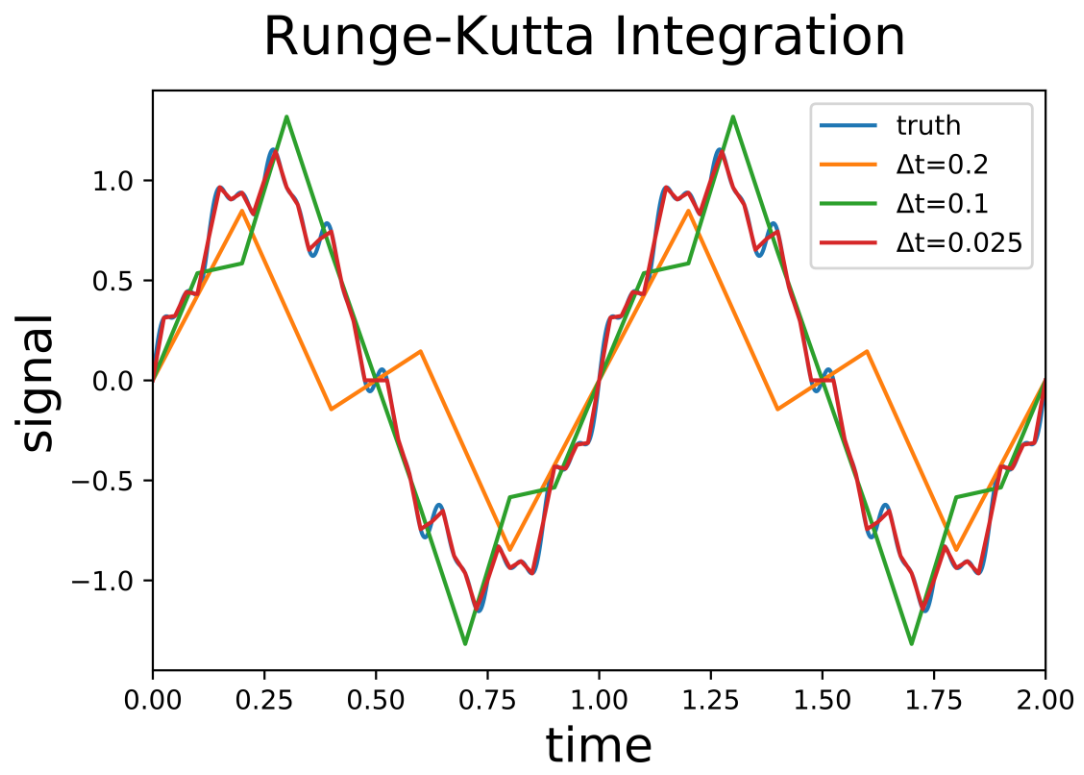
RK4 – The Classic Approach

- ‘Classic’, 4th order, it requires calculating the derivative FOUR times per iteration, expensive!

$$\dot{\boldsymbol{x}}_1 = \dot{\boldsymbol{x}}\{\boldsymbol{x}(t), t\}$$
$$\dot{\boldsymbol{x}}_2 = \dot{\boldsymbol{x}}\left\{\boldsymbol{x}(t) + \dot{\boldsymbol{x}}_1 \cdot \frac{\Delta t}{2}, t + \frac{\Delta t}{2}\right\}$$
$$\dot{\boldsymbol{x}}_3 = \dot{\boldsymbol{x}}\left\{\boldsymbol{x}(t) + \dot{\boldsymbol{x}}_2 \cdot \frac{\Delta t}{2}, t + \frac{\Delta t}{2}\right\}$$
$$\dot{\boldsymbol{x}}_4 = \dot{\boldsymbol{x}}\{\boldsymbol{x}(t) + \dot{\boldsymbol{x}}_3 \cdot \Delta t, t + \Delta t\}$$

$$\dot{\boldsymbol{x}}_{RK4} = \frac{1}{6}(\dot{\boldsymbol{x}}_1 + 2\dot{\boldsymbol{x}}_2 + 2\dot{\boldsymbol{x}}_3 + \dot{\boldsymbol{x}}_4)$$

But … it is stable and robust, hooray!



How does one decide on a numerical technique?

- Ask yourself some questions …

- Which method would work well for smooth dynamics, predictable from the past?
- Which method would work well for dynamics in which sudden changes can occur (e.g., sudden inputs)?
- Which method guarantees stability?
- Which method guarantees that it might not iterate forever?

PennState