Different Differential Equations with the Same Solution

Stephen Lucas* & James Sochacki

Department of Mathematics and Statistics
James Madison University, Harrisonburg VA

July 4, 2012

9th AIMS Conference on Dynamical Systems, Differential Equations and Applications
Special Session 39

Outline

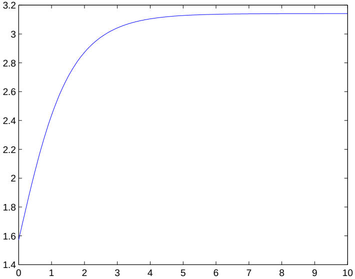
- What happens: different differential equations, different solutions
- Why it happens: error analysis
- Relations to the Power Series Method
- Symplectic solvers
- Effectively symplectic solver
- Future Work

Trig

Consider $y' = \sin y$ with $y(0) = \pi/2$.

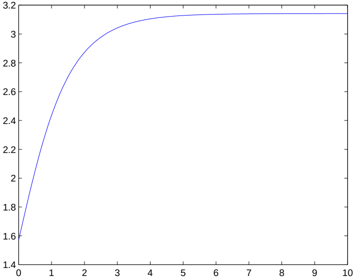
Trig

Consider $y' = \sin y$ with $y(0) = \pi/2$.
It has solution $y = 2tan^{-1}(e^t)$, which is far from obvious.



Trig

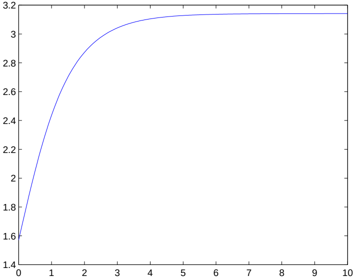
Consider $y' = \sin y$ with $y(0) = \pi/2$.
It has solution $y = 2tan^{-1}(e^t)$, which is far from obvious.



If we let $u_1 = y$, $u_2 = \sin(u_1)$ and $u_3 = \cos(u_2)$, then
$u_1' = u_2$, $u_1(0) = \pi/2$;

Trig

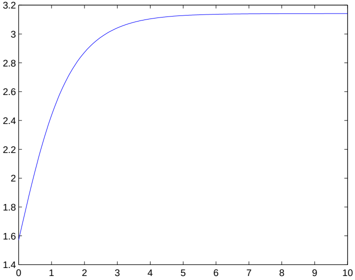
Consider $y' = \sin y$ with $y(0) = \pi/2$.
It has solution $y = 2tan^{-1}(e^t)$, which is far from obvious.



If we let $u_1 = y$, $u_2 = \sin(u_1)$ and $u_3 = \cos(u_2)$, then
$u_1' = u_2,\ u_1(0) = \pi/2;\ u_2' = u_2 u_3,\ u_2(0) = 1;$

Trig

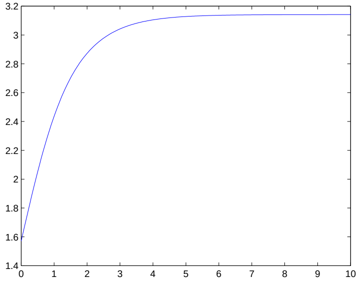
Consider $y' = \sin y$ with $y(0) = \pi/2$.
It has solution $y = 2tan^{-1}(e^t)$, which is far from obvious.



If we let $u_1 = y$, $u_2 = \sin(u_1)$ and $u_3 = \cos(u_2)$, then
$u_1' = u_2,\ u_1(0) = \pi/2;\ u_2' = u_2 u_3,\ u_2(0) = 1;\ u_3' = -u_2^2,\ u_3(0) = 0.$

Trig

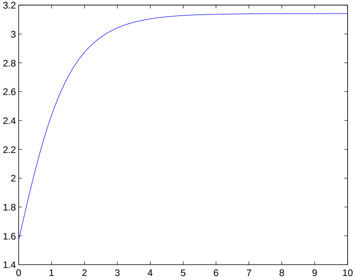
Consider $y' = \sin y$ with
$y(0) = \pi/2$.
It has solution
$y = 2tan^{-1}(e^t)$, which is far from obvious.



If we let $u_1 = y$, $u_2 = \sin(u_1)$ and $u_3 = \cos(u_2)$, then
$u_1' = u_2$, $u_1(0) = \pi/2$; $u_2' = u_2 u_3$, $u_2(0) = 1$; $u_3' = -u_2^2$, $u_3(0) = 0$.
While three first order odes may appear harder than one, compare two multiplications to evaluating a sine.

Trig

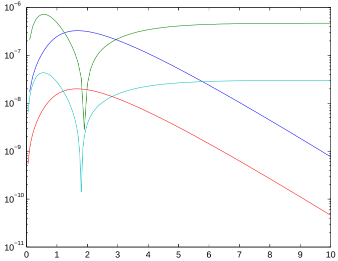
Consider $y' = \sin y$ with $y(0) = \pi/2$.
It has solution $y = 2tan^{-1}(e^t)$, which is far from obvious.



If we let $u_1 = y$, $u_2 = \sin(u_1)$ and $u_3 = \cos(u_2)$, then $u_1' = u_2$, $u_1(0) = \pi/2$; $u_2' = u_2 u_3$, $u_2(0) = 1$; $u_3' = -u_2^2$, $u_3(0) = 0$.
While three first order odes may appear harder than one, compare two multiplications to evaluating a sine. In a numerical solver, do we really need sine to machine precision?

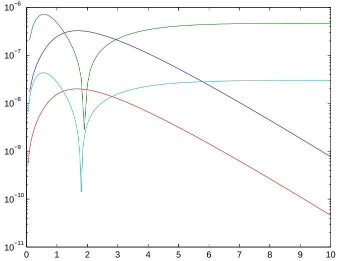
Trig Errors

Equal step RKO4 on $[0, 10]$ with 100, 200 intervals:

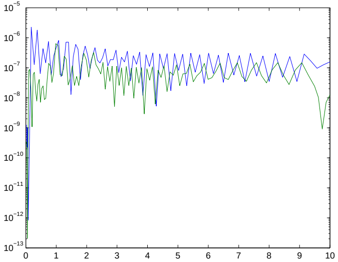


Trig Errors

Equal step RKO4 on $[0, 10]$ with 100, 200 intervals:



Matlab's ode45 on $[0, 10]$, absolute error 10^{-6}: 85 and 109 function evaluations.



PSM

$$u_1' = u_2,\ u_1(0) = \pi/2;\ u_2' = u_2 u_3,\ u_2(0) = 1;\ u_3' = -u_2^2,\ u_3(0) = 0.$$

PSM

$$u_1' = u_2,\ u_1(0) = \pi/2;\ u_2' = u_2 u_3,\ u_2(0) = 1;\ u_3' = -u_2^2,\ u_3(0) = 0.$$

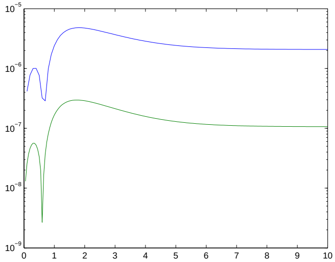
Power Series Method: at $t = 0$, replace variables by power series, explicitly find coefficients using Cauchy products and earlier coefficients (related to AD, Taylor methods).

PSM

$$u_1' = u_2,\ u_1(0) = \pi/2;\ u_2' = u_2 u_3,\ u_2(0) = 1;\ u_3' = -u_2^2,\ u_3(0) = 0.$$

Power Series Method: at $t = 0$, replace variables by power series, explicitly find coefficients using Cauchy products and earlier coefficients (related to AD, Taylor methods).

Equal step on $[0, 10]$ with 100,
200 intervals:



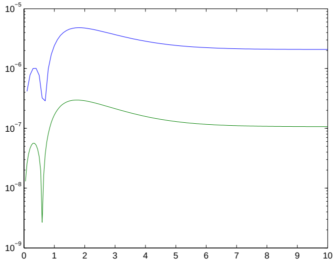
PSM

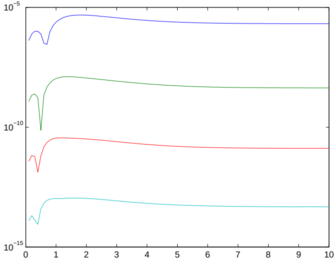
$$u_1' = u_2,\ u_1(0) = \pi/2;\ u_2' = u_2 u_3,\ u_2(0) = 1;\ u_3' = -u_2^2,\ u_3(0) = 0.$$

Power Series Method: at $t = 0$, replace variables by power series, explicitly find coefficients using Cauchy products and earlier coefficients (related to AD, Taylor methods).

Equal step on $[0, 10]$ with 100, 100 intervals, order 4, 6, 8, 10:

200 intervals:





Arbitrary Power

Consider $y' = y^{\alpha}$, $y(0) = y_0$,

Arbitrary Power

Consider $y' = y^\alpha$, $y(0) = y_0$, $y = \left(\left(t - \alpha t + y_0^{1-\alpha}\right)^{1/(\alpha-1)}\right)^{-1}$.

Arbitrary Power

Consider $y' = y^\alpha$, $y(0) = y_0$, $y = \left(\left(t - \alpha t + y_0^{1-\alpha}\right)^{1/(\alpha-1)}\right)^{-1}$.

With $u_1 = y$, $u_2 = y^\alpha$, $u_3 = 1/y$,

Arbitrary Power

Consider $y' = y^\alpha$, $y(0) = y_0$, $y = \left(\left(t - \alpha t + y_0^{1-\alpha}\right)^{1/(\alpha-1)}\right)^{-1}$.

With $u_1 = y$, $u_2 = y^\alpha$, $u_3 = 1/y$, we have $u_1' = y^\alpha = u_2$,

Arbitrary Power

Consider $y' = y^\alpha$, $y(0) = y_0$, $y = \left(\left(t - \alpha t + y_0^{1-\alpha}\right)^{1/(\alpha-1)}\right)^{-1}$.

With $u_1 = y$, $u_2 = y^\alpha$, $u_3 = 1/y$, we have $u_1' = y^\alpha = u_2$, $u_2' = \alpha y^{\alpha-1} y' = \alpha y^{2\alpha-1} = \alpha u_2^2 u_3$,

Arbitrary Power

Consider $y' = y^\alpha$, $y(0) = y_0$, $y = \left(\left(t - \alpha t + y_0^{1-\alpha}\right)^{1/(\alpha-1)}\right)^{-1}$.

With $u_1 = y$, $u_2 = y^\alpha$, $u_3 = 1/y$, we have $u_1' = y^\alpha = u_2$, $u_2' = \alpha y^{\alpha-1} y' = \alpha y^{2\alpha-1} = \alpha u_2^2 u_3$, $u_3' = -y^{-2} y' = -u_3^2 u_2$.

Arbitrary Power

Consider $y' = y^\alpha$, $y(0) = y_0$, $y = \left(\left(t - \alpha t + y_0^{1-\alpha}\right)^{1/(\alpha-1)}\right)^{-1}$.

With $u_1 = y$, $u_2 = y^\alpha$, $u_3 = 1/y$, we have $u_1' = y^\alpha = u_2$, $u_2' = \alpha y^{\alpha-1} y' = \alpha y^{2\alpha-1} = \alpha u_2^2 u_3$, $u_3' = -y^{-2} y' = -u_3^2 u_2$.

In addition, with $u_4 = u_2 u_3 = y^{\alpha-1}$,

Arbitrary Power

Consider $y' = y^\alpha$, $y(0) = y_0$, $y = \left(\left(t - \alpha t + y_0^{1-\alpha}\right)^{1/(\alpha-1)}\right)^{-1}$.

With $u_1 = y$, $u_2 = y^\alpha$, $u_3 = 1/y$, we have $u_1' = y^\alpha = u_2$, $u_2' = \alpha y^{\alpha-1} y' = \alpha y^{2\alpha-1} = \alpha u_2^2 u_3$, $u_3' = -y^{-2} y' = -u_3^2 u_2$.

In addition, with $u_4 = u_2 u_3 = y^{\alpha-1}$, $u_4' = (\alpha - 1) y^{\alpha-2} y'$ $= (\alpha - 1) y^{2\alpha-2} = (\alpha - 1) u_4^2$,

Arbitrary Power

Consider $y' = y^\alpha$, $y(0) = y_0$, $y = \left(\left(t - \alpha t + y_0^{1-\alpha}\right)^{1/(\alpha-1)}\right)^{-1}$.

With $u_1 = y$, $u_2 = y^\alpha$, $u_3 = 1/y$, we have $u_1' = y^\alpha = u_2$, $u_2' = \alpha y^{\alpha-1} y' = \alpha y^{2\alpha-1} = \alpha u_2^2 u_3$, $u_3' = -y^{-2} y' = -u_3^2 u_2$.

In addition, with $u_4 = u_2 u_3 = y^{\alpha-1}$, $u_4' = (\alpha-1)y^{\alpha-2}y'$ $= (\alpha-1)y^{2\alpha-2} = (\alpha-1)u_4^2$, so we could solve $u_1' = u_2$, $u_2' = \alpha u_2 u_4$, $u_3' = -u_3 u_4$, $u_4' = (\alpha-1)u_4^2$.

Arbitrary Power

Consider $y' = y^\alpha$, $y(0) = y_0$, $y = \left(\left(t - \alpha t + y_0^{1-\alpha}\right)^{1/(\alpha-1)}\right)^{-1}$.

With $u_1 = y$, $u_2 = y^\alpha$, $u_3 = 1/y$, we have $u_1' = y^\alpha = u_2$, $u_2' = \alpha y^{\alpha-1} y' = \alpha y^{2\alpha-1} = \alpha u_2^2 u_3$, $u_3' = -y^{-2} y' = -u_3^2 u_2$.

In addition, with $u_4 = u_2 u_3 = y^{\alpha-1}$, $u_4' = (\alpha-1) y^{\alpha-2} y'$ $= (\alpha-1) y^{2\alpha-2} = (\alpha-1) u_4^2$, so we could solve $u_1' = u_2$, $u_2' = \alpha u_2 u_4$, $u_3' = -u_3 u_4$, $u_4' = (\alpha-1) u_4^2$.

Or more simply $u_1' = u_1 u_4$, $u_4' = (\alpha-1) u_4^2$.

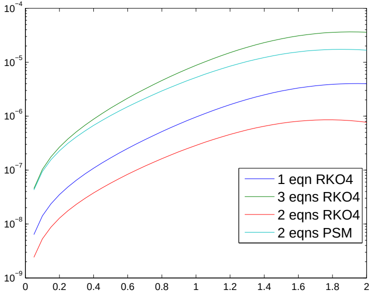
Arbitrary Power Errors

Set $\alpha = e/2 + i/\pi$, $y(0) = 1$, 40 intervals on $[0, 2]$, RKO4 with one, two, three equation versions, PSM on two equation version.

Arbitrary Power Errors

Set $\alpha = e/2 + i/\pi$, $y(0) = 1$, 40 intervals on $[0, 2]$, RKO4 with one, two, three equation versions, PSM on two equation version.

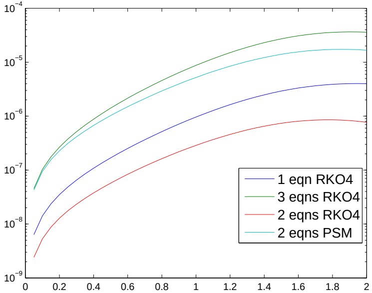
Error Magnitude



Arbitrary Power Errors

Set $\alpha = e/2 + i/\pi$, $y(0) = 1$, 40 intervals on $[0, 2]$, RKO4 with one, two, three equation versions, PSM on two equation version.

Error Magnitude



5th order PSM slightly better than RKO4 with two equations, twelfth order gives machine accuracy (or 100 intervals at eighth order).

Simple Pendulum

The simple pendulum is $\theta'' = -\sin\theta$ with $\theta(0) = \theta_0$, $\theta'(0) = 0$.

Simple Pendulum

The simple pendulum is $\theta'' = -\sin\theta$ with $\theta(0) = \theta_0$, $\theta'(0) = 0$. First order system: $\theta_1' = \theta_2$, $\theta_2' = -\sin\theta$ with $\theta_1(0) = \theta_0$, $\theta_2(0) = 0$.

Simple Pendulum

The simple pendulum is $\theta'' = -\sin\theta$ with $\theta(0) = \theta_0$, $\theta'(0) = 0$. First order system: $\theta_1' = \theta_2$, $\theta_2' = -\sin\theta$ with $\theta_1(0) = \theta_0$, $\theta_2(0) = 0$. Conservation of energy states that $(\theta')^2/2 - \cos\theta = C$.

Simple Pendulum

The simple pendulum is $\theta'' = -\sin\theta$ with $\theta(0) = \theta_0$, $\theta'(0) = 0$. First order system: $\theta_1' = \theta_2$, $\theta_2' = -\sin\theta$ with $\theta_1(0) = \theta_0$, $\theta_2(0) = 0$. Conservation of energy states that $(\theta')^2/2 - \cos\theta = C$.

No closed form solution.

Simple Pendulum

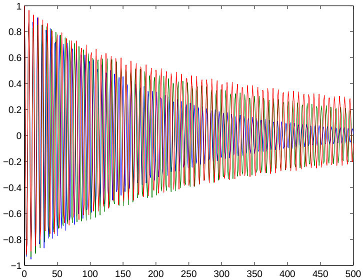
The simple pendulum is $\theta'' = -\sin\theta$ with $\theta(0) = \theta_0$, $\theta'(0) = 0$. First order system: $\theta_1' = \theta_2$, $\theta_2' = -\sin\theta$ with $\theta_1(0) = \theta_0$, $\theta_2(0) = 0$. Conservation of energy states that $(\theta')^2/2 - \cos\theta = C$.

No closed form solution.

Letting $u_1 = \theta$, $u_2 = \theta'$, $u_3 = \sin\theta$, $u_4 = \cos\theta$, then $u_1' = u_2$, $u_2' = -u_3$, $u_3' = u_2 u_4$, $u_4' = -u_2 u_3$ with $u_1(0) = \theta_0$, $u_2(0) = 0$, $u_3(0) = \sin(\theta_0)$, $u_4(0) = \cos(\theta_0)$.

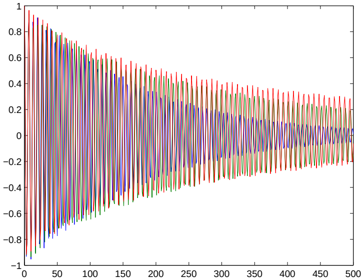
Pendulum Errors

500 intervals on $[0, 500]$ angle

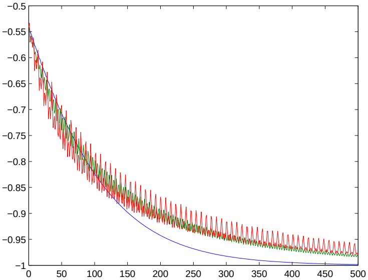


Pendulum Errors

500 intervals on $[0, 500]$ angle

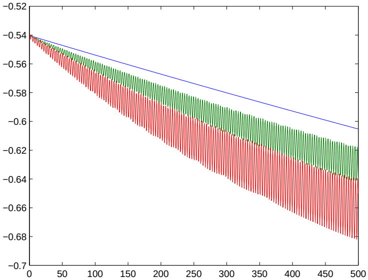


500 intervals on $[0, 500]$ energy



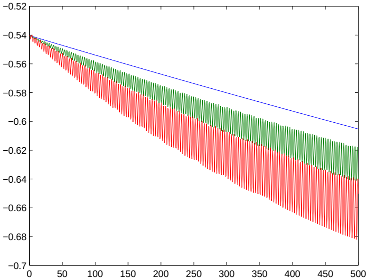
Pendulum Errors 2

1000 intervals on $[0, 500]$ energy



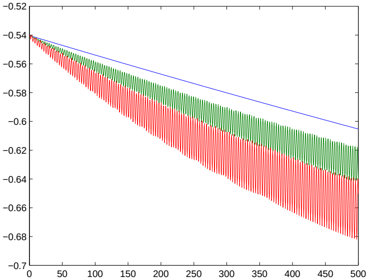
Pendulum Errors 2

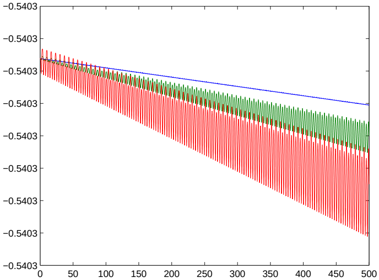
1000 intervals on $[0, 500]$ energy 10 000 intervals on $[0, 500]$



Pendulum Errors 2

1000 intervals on $[0, 500]$ energy 10 000 intervals on $[0, 500]$

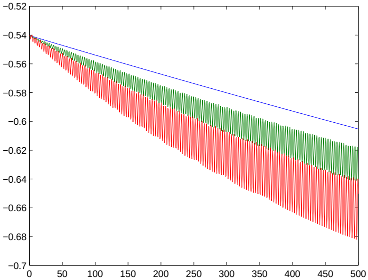


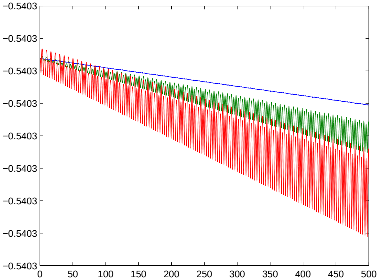


10 000 intervals on $[0, 500]$ $\Delta e = 2.9 \times 10^{-6}$.

Pendulum Errors 2

1000 intervals on $[0, 500]$ energy 10 000 intervals on $[0, 500]$





10 000 intervals on $[0, 500]$ $\Delta e = 2.9 \times 10^{-6}$. PSM with order 8 has $\Delta e = 2.2 \times 10^{-13}$, and order 12 energy is constant to machine precision.

Implications

- Runge-Kutta order 4 solutions to polynomial systems have different error from that of the original system, but function evaluations can be much faster.

Implications

- Runge-Kutta order 4 solutions to polynomial systems have different error from that of the original system, but function evaluations can be much faster. Not all differential equations are created equal.

Implications

- Runge-Kutta order 4 solutions to polynomial systems have different error from that of the original system, but function evaluations can be much faster. Not all differential equations are created equal.
- Power Series Method order 4 solutions usually have slightly more error than Runge-Kutta order 4 solutions for the same polynomial systems – and usually require more computational work!

Implications

- Runge-Kutta order 4 solutions to polynomial systems have different error from that of the original system, but function evaluations can be much faster. Not all differential equations are created equal.
- Power Series Method order 4 solutions usually have slightly more error than Runge-Kutta order 4 solutions for the same polynomial systems – and usually require more computational work!
- But, the PSM can be made of arbitrary order, and has many other advantages...

Error Analysis, Initial Idea

Euler's method for $y' = f(t, y(t))$ is $y_{n+1} = y_n + h\,f(t_n, y_n)$ with local error $O(h^2)$ and global error $O(h)$.

Error Analysis, Initial Idea

Euler's method for $y' = f(t, y(t))$ is $y_{n+1} = y_n + h\,f(t_n, y_n)$ with local error $O(h^2)$ and global error $O(h)$. Typical derivation via Taylor series: $y(t_0 + h) = y(t_0) + hy'(t_0) + \frac{h^2}{2}y''(t_0) + O(h^3)$. Replace $y'(t_0)$ by $f(t_0, y(t_0))$, error is $h^2y''(t_0)/2 + O(h^3)$, or in Lagrange form $h^2y''(\xi)/2$, $\xi \in [t_0, t_0 + h]$.

Error Analysis, Initial Idea

Euler's method for $y' = f(t, y(t))$ is $y_{n+1} = y_n + h\, f(t_n, y_n)$ with local error $O(h^2)$ and global error $O(h)$. Typical derivation via Taylor series: $y(t_0 + h) = y(t_0) + hy'(t_0) + \frac{h^2}{2} y''(t_0) + O(h^3)$. Replace $y'(t_0)$ by $f(t_0, y(t_0))$, error is $h^2 y''(t_0)/2 + O(h^3)$, or in Lagrange form $h^2 y''(\xi)/2$, $\xi \in [t_0, t_0 + h]$.

Higher order derivations almost always stop here, and leave the impression that error is proportional to h and depends on derivatives of y.

Error Analysis, Initial Idea

Euler's method for $y' = f(t, y(t))$ is $y_{n+1} = y_n + h\,f(t_n, y_n)$ with local error $O(h^2)$ and global error $O(h)$. Typical derivation via Taylor series: $y(t_0 + h) = y(t_0) + hy'(t_0) + \frac{h^2}{2}y''(t_0) + O(h^3)$. Replace $y'(t_0)$ by $f(t_0, y(t_0))$, error is $h^2y''(t_0)/2 + O(h^3)$, or in Lagrange form $h^2y''(\xi)/2$, $\xi \in [t_0, t_0 + h]$.

Higher order derivations almost always stop here, and leave the impression that error is proportional to h and depends on derivatives of y. But (for Euler), since $y''(\xi) = \frac{\partial f}{\partial t}(\xi, y(\xi)) + \frac{\partial f}{\partial y}(\xi, y(\xi))f(\xi, y(\xi))$, we can relate the error to the RHS.

Error Analysis, Trig

If $y_i' = f_i(t, y_1(t), y_2(t), \ldots, y_n(t))$ for $i = 1, 2, \ldots, n$, errors are $h^2 y_i''(\xi_i)/2$ where $y_i''(\xi_i) = \frac{\partial f_i}{\partial t} + \sum_{j=1}^{n} \frac{\partial f_i}{\partial y_j} f_j$.

Error Analysis, Trig

If $y_i' = f_i(t, y_1(t), y_2(t), \ldots, y_n(t))$ for $i = 1, 2, \ldots, n$, errors are $h^2 y_i''(\xi_i)/2$ where $y_i''(\xi_i) = \frac{\partial f_i}{\partial t} + \sum_{j=1}^{n} \frac{\partial f_i}{\partial y_j} f_j$.

$y' = \sin y$, error $\frac{h}{2} y''(\xi) = \frac{h}{2} \cos(\xi) \sin(\xi)$

Error Analysis, Trig

If $y_i' = f_i(t, y_1(t), y_2(t), \ldots, y_n(t))$ for $i = 1, 2, \ldots, n$, errors are $h^2 y_i''(\xi_i)/2$ where $y_i''(\xi_i) = \frac{\partial f_i}{\partial t} + \sum_{j=1}^{n} \frac{\partial f_i}{\partial y_j} f_j$.

$y' = \sin y$, error $\frac{h}{2} y''(\xi) = \frac{h}{2} \cos(\xi) \sin(\xi)$

$[u_1 = y,\ u_2 = \sin y,\ u_3 = \cos y]$: $u_1' = u_2,\ u_2' = u_2 u_3,\ u_3' = -u_2^2$,

Error Analysis, Trig

If $y_i' = f_i(t, y_1(t), y_2(t), \dots, y_n(t))$ for $i = 1, 2, \dots, n$, errors are $h^2 y_i''(\xi_i)/2$ where $y_i''(\xi_i) = \frac{\partial f_i}{\partial t} + \sum_{j=1}^{n} \frac{\partial f_i}{\partial y_j} f_j$.

$y' = \sin y$, error $\frac{h}{2} y''(\xi) = \frac{h}{2} \cos(\xi) \sin(\xi)$

$[u_1 = y,\ u_2 = \sin y,\ u_3 = \cos y]$: $u_1' = u_2,\ u_2' = u_2 u_3,\ u_3' = -u_2^2$, u_1 error $\frac{h}{2} u_2(\xi_1) u_3(\xi_1)$, u_2 error $\frac{h}{2}\left(u_2^2(\xi_2) u_3(\xi_2) - u_2^3(\xi_2)\right)$, u_3 error $-h u_2^2(\xi_3) u_3(\xi_3)$, $t_0 \le \xi_i \le t_0 + h$.

Error Analysis, Trig

If $y_i' = f_i(t, y_1(t), y_2(t), \ldots, y_n(t))$ for $i = 1, 2, \ldots, n$, errors are $h^2 y_i''(\xi_i)/2$ where $y_i''(\xi_i) = \frac{\partial f_i}{\partial t} + \sum_{j=1}^{n} \frac{\partial f_i}{\partial y_j} f_j$.

$y' = \sin y$, error $\frac{h}{2} y''(\xi) = \frac{h}{2} \cos(\xi) \sin(\xi)$

$[u_1 = y,\ u_2 = \sin y,\ u_3 = \cos y]$: $u_1' = u_2$, $u_2' = u_2 u_3$, $u_3' = -u_2^2$, u_1 error $\frac{h}{2} u_2(\xi_1) u_3(\xi_1)$, u_2 error $\frac{h}{2}\left(u_2^2(\xi_2) u_3(\xi_2) - u_2^3(\xi_2)\right)$, u_3 error $-h u_2^2(\xi_3) u_3(\xi_3)$, $t_0 \le \xi_i \le t_0 + h$.

Error in u_1 is $\dfrac{h}{2}(\sin(\xi_1)\cos(\xi_1)) + O(h^2)$.

Implications of Polynomial Form

- Rewriting a system of differential equations in polynomial form introduces additional variables.

Implications of Polynomial Form

- Rewriting a system of differential equations in polynomial form introduces additional variables.
- Using an *n*th order solver introduces $O(h^n)$ errors in these terms, which contribute to the $O(h^n)$ error in the variables corresponding to those in the original system.

Implications of Polynomial Form

- Rewriting a system of differential equations in polynomial form introduces additional variables.
- Using an nth order solver introduces $O(h^n)$ errors in these terms, which contribute to the $O(h^n)$ error in the variables corresponding to those in the original system.
- The final error is $O(h^n)$, but the proportionality constant will change. It may increase or decrease depending on the magnitude and sign of the introduced errors.

Implications of Polynomial Form

- Rewriting a system of differential equations in polynomial form introduces additional variables.
- Using an nth order solver introduces $O(h^n)$ errors in these terms, which contribute to the $O(h^n)$ error in the variables corresponding to those in the original system.
- The final error is $O(h^n)$, but the proportionality constant will change. It may increase or decrease depending on the magnitude and sign of the introduced errors.
- Polynomial form is superior because the rhs is much more efficient to evaluate – compare a few of multiplications to evaluating transcendental functions.

Implications of Polynomial Form

- Rewriting a system of differential equations in polynomial form introduces additional variables.
- Using an nth order solver introduces $O(h^n)$ errors in these terms, which contribute to the $O(h^n)$ error in the variables corresponding to those in the original system.
- The final error is $O(h^n)$, but the proportionality constant will change. It may increase or decrease depending on the magnitude and sign of the introduced errors.
- Polynomial form is superior because the rhs is much more efficient to evaluate – compare a few of multiplications to evaluating transcendental functions.
- Every system of odes can be rewritten in polynomial form in an algorithmic manner.

Implications of Polynomial Form

- Rewriting a system of differential equations in polynomial form introduces additional variables.
- Using an nth order solver introduces $O(h^n)$ errors in these terms, which contribute to the $O(h^n)$ error in the variables corresponding to those in the original system.
- The final error is $O(h^n)$, but the proportionality constant will change. It may increase or decrease depending on the magnitude and sign of the introduced errors.
- Polynomial form is superior because the rhs is much more efficient to evaluate – compare a few of multiplications to evaluating transcendental functions.
- Every system of odes can be rewritten in polynomial form in an algorithmic manner. Functions like $(e^t - 1)/t$ and $(\sin t)/t$ at $t = 0$ can cause problems.

Implications of Power Series Method

In all cases so far, PSM order 4 inferior to Runge-Kutta order 4, in terms of error and computational effort per step.

Implications of Power Series Method

In all cases so far, PSM order 4 inferior to Runge-Kutta order 4, in terms of error and computational effort per step.

PSM generates the exact Taylor series of order n expanding at a given point. Runge-Kutta matches the initial Taylor coefficients, but has some additional contributions.

Implications of Power Series Method

In all cases so far, PSM order 4 inferior to Runge-Kutta order 4, in terms of error and computational effort per step.

PSM generates the exact Taylor series of order n expanding at a given point. Runge-Kutta matches the initial Taylor coefficients, but has some additional contributions.

Runge-Kutta is equivalent to an infinite power series, and the approximate tail is usually better than none at all.

Implications of Power Series Method

In all cases so far, PSM order 4 inferior to Runge-Kutta order 4, in terms of error and computational effort per step.

PSM generates the exact Taylor series of order n expanding at a given point. Runge-Kutta matches the initial Taylor coefficients, but has some additional contributions.

Runge-Kutta is equivalent to an infinite power series, and the approximate tail is usually better than none at all.

Computationally, each multiplication in Runge-Kutta requires Cauchy products in PSM:

$$\left(\sum_{i=0}^{\infty} a_i x^i\right)\left(\sum_{i=0}^{\infty} b_i x^i\right) = \sum_{i=0}^{\infty}\left(\sum_{j=0}^{i} a_j b_{i-j}\right) x^i.$$

Why PSM

- Every system of odes can be rewritten in polynomial form algorithmically. Every analytic function can be equivalently replaced by a system of odes. Every system of polynomial odes has an equivalent quadratic system of odes.

Why PSM

- Every system of odes can be rewritten in polynomial form algorithmically. Every analytic function can be equivalently replaced by a system of odes. Every system of polynomial odes has an equivalent quadratic system of odes.
- A wide range of modeling problems are polynomial odes.

Why PSM

- Every system of odes can be rewritten in polynomial form algorithmically. Every analytic function can be equivalently replaced by a system of odes. Every system of polynomial odes has an equivalent quadratic system of odes.
- A wide range of modeling problems are polynomial odes.
- Arbitrary order available automatically or adaptively. Can balance order versus number of intervals to minimize.

Why PSM

- Every system of odes can be rewritten in polynomial form algorithmically. Every analytic function can be equivalently replaced by a system of odes. Every system of polynomial odes has an equivalent quadratic system of odes.
- A wide range of modeling problems are polynomial odes.
- Arbitrary order available automatically or adaptively. Can balance order versus number of intervals to minimize.
- A priori error estimate available.

Why PSM (cont'd)

- Numerical solution (as a power series) is available for all time, not just at data points (compare with extrapolation methods).

Why PSM (cont'd)

- Numerical solution (as a power series) is available for all time, not just at data points (compare with extrapolation methods).
  - End condition is $g(t, y(t)) = 0$.

Why PSM (cont'd)

- Numerical solution (as a power series) is available for all time, not just at data points (compare with extrapolation methods).
  - End condition is $g(t, y(t)) = 0$.
  - Even better, PSM can be used to identify crossing times when differential equation has discontinuities, like $x'' + \mu x^+ - \nu x^- = 0$, $\mu, \nu \geq 0$, $x^+ = \max\{x, 0\}$, $x^- = \max\{-x, 0\}$.

Why PSM (cont'd)

- Numerical solution (as a power series) is available for all time, not just at data points (compare with extrapolation methods).
    - End condition is $g(t, y(t)) = 0$.
    - Even better, PSM can be used to identify crossing times when differential equation has discontinuities, like $x'' + \mu x^+ - \nu x^- = 0$, $\mu, \nu \geq 0$, $x^+ = \max\{x, 0\}$, $x^- = \max\{-x, 0\}$.
    - Makes delay differential equations trivial to solve.

Why PSM (cont'd)

- Numerical solution (as a power series) is available for all time, not just at data points (compare with extrapolation methods).
  - End condition is $g(t, y(t)) = 0$.
  - Even better, PSM can be used to identify crossing times when differential equation has discontinuities, like $x'' + \mu x^+ - \nu x^- = 0$, $\mu, \nu \geq 0$, $x^+ = \max\{x, 0\}$, $x^- = \max\{-x, 0\}$.
  - Makes delay differential equations trivial to solve.
- Arbitrary order means we can easily push to machine accuracy solutions, regardless of the required precision. For Hamiltonian systems of odes, we can conserve energy to machine precision – effectively symplectic.

Symplectic Methods

A symplectic method numerically approximates a system of odes in such a way that a first integral, or Hamiltonian, is conserved.

Symplectic Methods

A symplectic method numerically approximates a system of odes in such a way that a first integral, or Hamiltonian, is conserved.

Leapfrog integration (Feynman Lectures): To solve $x' = v$, $v' = f(x)$, $x_i = x_{i-1} + v_{i-1/2}\Delta t$, $v_{i+1/2} = v_{i-1/2} + f(x_i)\Delta t$,

Symplectic Methods

A symplectic method numerically approximates a system of odes in such a way that a first integral, or Hamiltonian, is conserved.

Leapfrog integration (Feynman Lectures): To solve $x' = v$, $v' = f(x)$, $x_i = x_{i-1} + v_{i-1/2}\Delta t$, $v_{i+1/2} = v_{i-1/2} + f(x_i)\Delta t$, or $x_{i+1} = x_i + v_i\Delta t + \frac{f(x_i)}{2}\Delta t^2$, $v_{i+1} = v_i + \frac{1}{2}(f(x_i) + f(x_{i+1})\Delta t$.

Symplectic Methods

A symplectic method numerically approximates a system of odes in such a way that a first integral, or Hamiltonian, is conserved.

Leapfrog integration (Feynman Lectures): To solve $x' = v$, $v' = f(x)$, $x_i = x_{i-1} + v_{i-1/2}\Delta t$, $v_{i+1/2} = v_{i-1/2} + f(x_i)\Delta t$, or $x_{i+1} = x_i + v_i\Delta t + \frac{f(x_i)}{2}\Delta t^2$, $v_{i+1} = v_i + \frac{1}{2}(f(x_i) + f(x_{i+1})\Delta t$. It is invariant under time reversal, and "area preserving" in position/momentum space, and energy is nearly conserved (periodic).

Symplectic Methods

A symplectic method numerically approximates a system of odes in such a way that a first integral, or Hamiltonian, is conserved.

Leapfrog integration (Feynman Lectures): To solve $x' = v$, $v' = f(x)$, $x_i = x_{i-1} + v_{i-1/2}\Delta t$, $v_{i+1/2} = v_{i-1/2} + f(x_i)\Delta t$, or $x_{i+1} = x_i + v_i\Delta t + \frac{f(x_i)}{2}\Delta t^2$, $v_{i+1} = v_i + \frac{1}{2}(f(x_i) + f(x_{i+1})\Delta t$. It is invariant under time reversal, and "area preserving" in position/momentum space, and energy is nearly conserved (periodic).

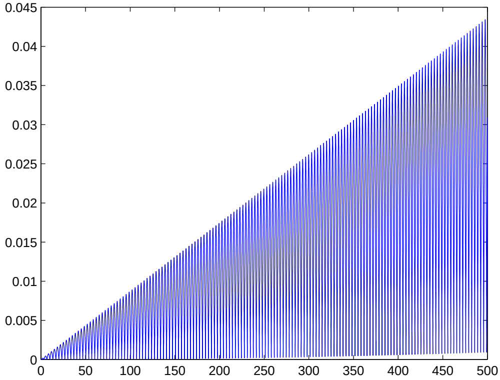
Forest-Ruth (1990) is fourth order: Given x_i, v_i: $x_a = x_i + \theta h v_i/2$, $v_a = v_i + \theta h f(x_a)$, $x_b = x_a + (1-\theta)hv_a/2$, $v_b = v_a + (1-2\theta)hf(x_b)$, $x_c = x_b + (1-\theta)hv_b/2$, $v_{i+1} = v_b + \theta h f(x_c)$, $x_{i+1} = x_c + \theta h v_{i+1}/2$ with $\theta = 1/(2-\sqrt[3]{2}) \approx 1.35$.

Symplectic Simple Pendulum

10 000 intervals on $[0, 500]$: energy periodic, range 2.6×10^{-4}.

Symplectic Simple Pendulum

10 000 intervals on $[0, 500]$: energy periodic, range 2.6×10^{-4}.



But actual error:

Symplectic Issues

- More accurate fourth order require more function evaluations, or are implicit.

Symplectic Issues

- More accurate fourth order require more function evaluations, or are implicit.
- Symplectic solvers for more complicated Hamiltonians can be very complicated.

Symplectic Issues

- More accurate fourth order require more function evaluations, or are implicit.
- Symplectic solvers for more complicated Hamiltonians can be very complicated.
- (Apparently,) all symplectic algorithms require equal step sizes.

Symplectic Issues

- More accurate fourth order require more function evaluations, or are implicit.
- Symplectic solvers for more complicated Hamiltonians can be very complicated.
- (Apparently,) all symplectic algorithms require equal step sizes.
- Symplectic methods still have error. Since it is not in the "energy," it is in the "phase."

Effectively Symplectic

We call a numerical method “effectively symplectic” if the energy is conserved without explicitly using the Hamiltonian form.

Effectively Symplectic

We call a numerical method “effectively symplectic” if the energy is conserved without explicitly using the Hamiltonian form. Any numerical method that gives the true solution to machine precision will conserve energy to machine precision, and hence be effectively symplectic.

Effectively Symplectic

We call a numerical method “effectively symplectic” if the energy is conserved without explicitly using the Hamiltonian form. Any numerical method that gives the true solution to machine precision will conserve energy to machine precision, and hence be effectively symplectic. The PSM or any other related method can easily be effectively symplectic, and is adaptive

Effectively Symplectic

We call a numerical method "effectively symplectic" if the energy is conserved without explicitly using the Hamiltonian form. Any numerical method that gives the true solution to machine precision will conserve energy to machine precision, and hence be effectively symplectic. The PSM or any other related method can easily be effectively symplectic, and is adaptive

- Simple pendulum example: order twelve, 10 000 intervals on $[0, 500]$.

Effectively Symplectic

We call a numerical method "effectively symplectic" if the energy is conserved without explicitly using the Hamiltonian form. Any numerical method that gives the true solution to machine precision will conserve energy to machine precision, and hence be effectively symplectic. The PSM or any other related method can easily be effectively symplectic, and is adaptive

- Simple pendulum example: order twelve, 10 000 intervals on $[0, 500]$.
- Pruett, Ingham & Herman (2011): An adaptive and parallel implementation of the PSM method for the N-body problem, currently fastest accurate solver for such problems.

Effectively Symplectic

We call a numerical method “effectively symplectic” if the energy is conserved without explicitly using the Hamiltonian form. Any numerical method that gives the true solution to machine precision will conserve energy to machine precision, and hence be effectively symplectic. The PSM or any other related method can easily be effectively symplectic, and is adaptive

- Simple pendulum example: order twelve, 10 000 intervals on $[0, 500]$.
- Pruett, Ingham & Herman (2011): An adaptive and parallel implementation of the PSM method for the N-body problem, currently fastest accurate solver for such problems.
- Double pendulum: later today!

Future Work

- Using a priori error bound and requested error, identify most efficient combination of order and interval width for PSM.

Future Work

- Using a priori error bound and requested error, identify most efficient combination of order and interval width for PSM.
- Automatically converting a system to polynomial form, minimizing of number of Cauchy products (quadratic terms on rhs) using "intermediate variables."

Future Work

- Using a priori error bound and requested error, identify most efficient combination of order and interval width for PSM.
- Automatically converting a system to polynomial form, minimizing of number of Cauchy products (quadratic terms on rhs) using "intermediate variables."
- Automatically dealing with "difficult" functions like $(\sin u(t))/u(t)$ or $(e^{u(t)} - 1)/u(t)$ near $u(t) = 0$.

Future Work

- Using a priori error bound and requested error, identify most efficient combination of order and interval width for PSM.
- Automatically converting a system to polynomial form, minimizing of number of Cauchy products (quadratic terms on rhs) using "intermediate variables."
- Automatically dealing with "difficult" functions like $(\sin u(t))/u(t)$ or $(e^{u(t)} - 1)/u(t)$ near $u(t) = 0$.
- A more careful survey of symplectic methods, and comparison with the PSM.

Future Work

- Using a priori error bound and requested error, identify most efficient combination of order and interval width for PSM.
- Automatically converting a system to polynomial form, minimizing of number of Cauchy products (quadratic terms on rhs) using “intermediate variables.”
- Automatically dealing with “difficult” functions like $(\sin u(t))/u(t)$ or $(e^{u(t)} - 1)/u(t)$ near $u(t) = 0$.
- A more careful survey of symplectic methods, and comparison with the PSM.
- Implementation for delay differential equations, piecewise functions, ...

Future Work

- Using a priori error bound and requested error, identify most efficient combination of order and interval width for PSM.
- Automatically converting a system to polynomial form, minimizing of number of Cauchy products (quadratic terms on rhs) using "intermediate variables."
- Automatically dealing with "difficult" functions like $(\sin u(t))/u(t)$ or $(e^{u(t)} - 1)/u(t)$ near $u(t) = 0$.
- A more careful survey of symplectic methods, and comparison with the PSM.
- Implementation for delay differential equations, piecewise functions, ...

Thank You

JAMES MADISON UNIVERSITY