

- 1) know $\{y\}$ \rightarrow pos AND val
- 2) use $\{y\}$ in dynamic equations to determine $\{y\}$ \rightarrow vel AND acc
- 3) use $\{y\}$ to predict the future

Forward Time Integration of First-Order Initial-Value Problems

$$\{y\} = f(\{y\}, t)$$

Euler's Method

one function eval per step

given first-order initial-value differential equations of the form $\{y\} = f(\{y\}, t)$

must know $\{y_i\}$ at time t_i for time step $h = t_{i+1} - t_i$

determine the slope at the beginning of the interval $k = f(y_i, t_i)$

$$y_{i+1} = y_i + k h$$

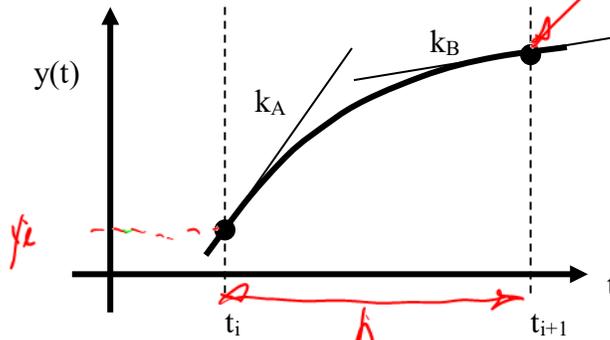
$$\{y_{i+1}\} = \{y_i\} + \{k_i\} h$$

Second Order Runge-Kutta

two function evals per step

given first-order initial-value differential equations of the form $\{y\} = f(\{y\}, t)$

must know $\{y_i\}$ at time t_i for time step $h = t_{i+1} - t_i$



determine the slope at the beginning of the interval $k_A = f(y_i, t_i)$

estimate the slope at the end of the interval using Euler's method $k_B = f(y_i + k_A h, t_i + h)$

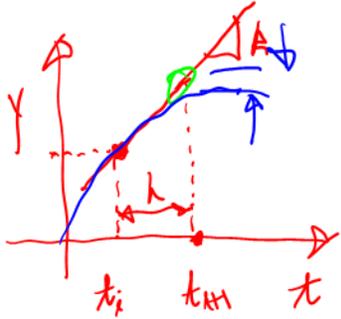
use the mean of these two slopes $k_M = \frac{1}{2}(k_A + k_B)$

$$y_{i+1} = y_i + k_M h$$

slope k
 evaluate $\{y\}$ from $\{y\}$
 # of function evaluation to calculate $\{y\}$

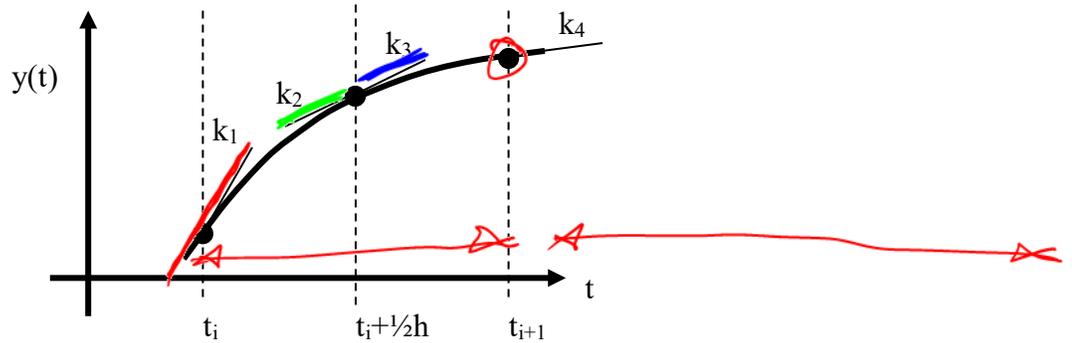
Euler error

use differential eqns twice



Fourth Order Runge-Kutta (RK4)

four function evaluation per step



- 1) variable time step
- 2) do not need prior values before t_i

determine the slope at the beginning of the interval $k_1 = f(y_i, t_i)$ *Euler*

estimate the slope at the midpoint of the interval using Euler's method $k_2 = f(y_i + k_1 \frac{h}{2}, t_i + \frac{h}{2})$ *half step*

estimate the slope at the midpoint of the interval using the new value $k_3 = f(y_i + k_2 \frac{h}{2}, t_i + \frac{h}{2})$ *half step*

estimate the slope at the end of the interval using the newest value $k_4 = f(y_i + k_3 h, t_i + h)$ *full step*

use a weighted mean of slopes $k_M = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$

$$y_{i+1} = y_i + k_M h$$

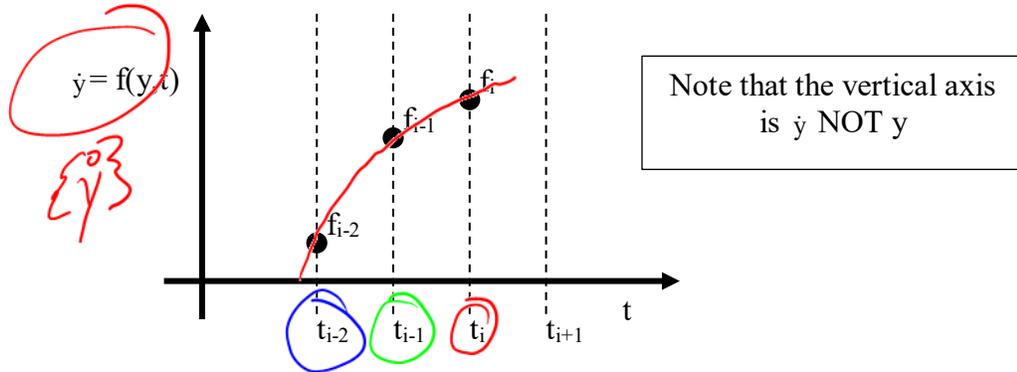
fourth order accuracy

Second-degree (Quadratic) Predictor and Third-degree (Cubic) Corrector

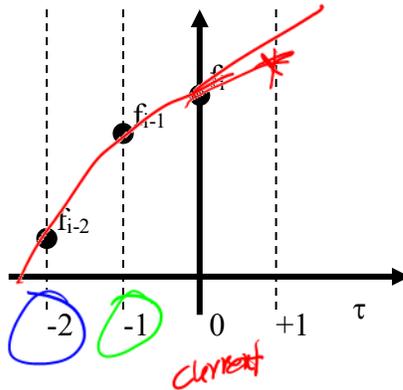
PECE

given first-order initial-value differential equations of the form $\dot{y} = f(y, t)$

must know $\{y\}_i, \{y\}_{i-1}, \{y\}_{i-2}$ respectively at times t_i, t_{i-1} and t_{i-2}
 may determine $f_i = \{y\}'_i, f_{i-1} = \{y\}'_{i-1}, f_{i-2} = \{y\}'_{i-2}$



for fixed time step h $\tau = (t - t_i) / h$ $d\tau = dt / h$ $dt = h d\tau$



fixed time step need all values prior to t_i

use second-degree polynomial predictor $f = a_0 + a_1 \tau + a_2 \tau^2 = \begin{bmatrix} 1 & \tau & \tau^2 \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \end{Bmatrix}$

$$\begin{Bmatrix} f_i \\ f_{i-1} \\ f_{i-2} \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & -1 & 1 \\ 1 & -2 & 4 \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \end{Bmatrix}$$

$$\begin{Bmatrix} a_0 \\ a_1 \\ a_2 \end{Bmatrix} = \begin{bmatrix} 2 & 0 & 0 \\ 3 & -4 & 1 \\ 1 & -2 & 1 \end{bmatrix} \begin{Bmatrix} f_i \\ f_{i-1} \\ f_{i-2} \end{Bmatrix} / 2$$

know coefficients a_0, a_1 and a_2 for interpolant $f(\tau) = a_0 + a_1 \tau + a_2 \tau^2 = \dot{y}(\tau)$

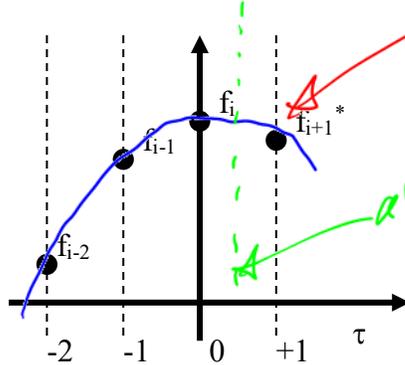
$$y_{i+1} = \int_{t_i}^{t_{i+1}} \dot{y} dt + y_i = h \int_0^1 f d\tau + y_i = h (a_0\tau + a_1\tau^2/2 + a_2\tau^3/3) \Big|_0^1 + y_i$$

$$y_{i+1} = y_i + h (a_0 + a_1/2 + a_2/3)$$

PREDICTOR (3-step Adams-Bashforth)

$$y_{i+1} = y_i + h(23 f_i - 16 f_{i-1} + 5 f_{i-2})/12$$

now use predicted value of y_{i+1} to compute $f(y_{i+1}) = f_{i+1}^*$



forward from f_i to t_{i+1}

one function evaluation per step

$y_{i+1} = y_i + h f_i$
third order

use third-degree polynomial corrector

$$f = b_0 + b_1 \tau + b_2 \tau^2 + b_3 \tau^3 = [1 \ \tau \ \tau^2 \ \tau^3] \begin{Bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{Bmatrix}$$

coefficients

$$\begin{Bmatrix} f_{i+1}^* \\ f_i \\ f_{i-1} \\ f_{i-2} \end{Bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & -1 \\ 1 & -2 & 4 & -8 \end{bmatrix} \begin{Bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{Bmatrix} \quad \begin{Bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{Bmatrix} = \begin{bmatrix} 0 & 6 & 0 & 0 \\ 2 & 3 & -6 & 1 \\ 3 & -6 & 3 & 0 \\ 1 & -3 & 3 & -1 \end{bmatrix} \begin{Bmatrix} f_{i+1}^* \\ f_i \\ f_{i-1} \\ f_{i-2} \end{Bmatrix} / 6$$

know coefficients b_0, b_1, b_2 and b_3 for new interpolant

$$f(\tau) = b_0 + b_1 \tau + b_2 \tau^2 + b_3 \tau^3 = \dot{y}(\tau)$$

$$y_{i+1} = \int_{t_i}^{t_{i+1}} \dot{y} dt + y_i = h \int_0^1 f d\tau + y_i = h (b_0\tau + b_1\tau^2/2 + b_2\tau^3/3 + b_3\tau^4/4) \Big|_0^1 + y_i$$

$$y_{i+1} = y_i + h (b_0 + b_1/2 + b_2/3 + b_3/4)$$

CORRECTOR (4-step Adams-Moulton)

$$y_{i+1} = y_i + h (9 f_{i+1}^* + 19 f_i - 5 f_{i-1} + f_{i-2}) / 24$$

fourth order

1) fewer function evaluation → faster
2) compare y_{i+1} predictor-only to y_{i+1} from pred-corr

integrate $\int_{t_i}^{t_{i+1}} \dot{y} dt$

variable time step second-degree polynomial predictor for f $f = a_0 + a_1 (t-t_i) + a_2 (t-t_i)^2$

$$\begin{Bmatrix} f_i \\ f_{i-1} \\ f_{i-2} \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & (t_{i-1} - t_i) & (t_{i-1} - t_i)^2 \\ 1 & (t_{i-2} - t_i) & (t_{i-2} - t_i)^2 \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \end{Bmatrix} = [A] \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \end{Bmatrix}$$

$$\begin{Bmatrix} a_0 \\ a_1 \\ a_2 \end{Bmatrix} = [A]^{-1} \begin{Bmatrix} f_i \\ f_{i-1} \\ f_{i-2} \end{Bmatrix}$$

$$y_{i+1} = \int_{t_i}^{t_{i+1}} \dot{y} dt + y_i$$

PREDICTOR

$$y_{i+1} = y_i + a_0 (t_{i+1}-t_i) + a_1 (t_{i+1}-t_i)^2/2 + a_2 (t_{i+1}-t_i)^3/3$$

use predicted value of y_{i+1} to compute $f(y_{i+1}, t) = f_{i+1}^*$

use third-degree polynomial corrector for f $f = b_0 + b_1 (t-t_i) + b_2 (t-t_i)^2 + b_3 (t-t_i)^3$

$$\begin{Bmatrix} f_{i+1}^* \\ f_i \\ f_{i-1} \\ f_{i-2} \end{Bmatrix} = \begin{bmatrix} 1 & (t_{i+1} - t_i) & (t_{i+1} - t_i)^2 & (t_{i+1} - t_i)^3 \\ 1 & 0 & 0 & 0 \\ 1 & (t_{i-1} - t_i) & (t_{i-1} - t_i)^2 & (t_{i-1} - t_i)^3 \\ 1 & (t_{i-2} - t_i) & (t_{i-2} - t_i)^2 & (t_{i-2} - t_i)^3 \end{bmatrix} \begin{Bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{Bmatrix} = [B] \begin{Bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{Bmatrix}$$

$$\begin{Bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{Bmatrix} = [B]^{-1} \begin{Bmatrix} f_{i+1}^* \\ f_i \\ f_{i-1} \\ f_{i-2} \end{Bmatrix}$$

CORRECTOR

$$y_{i+1} = y_i + b_0 (t_{i+1}-t_i) + b_1 (t_{i+1}-t_i)^2/2 + b_2 (t_{i+1}-t_i)^3/3 + b_3 (t_{i+1}-t_i)^4/4$$

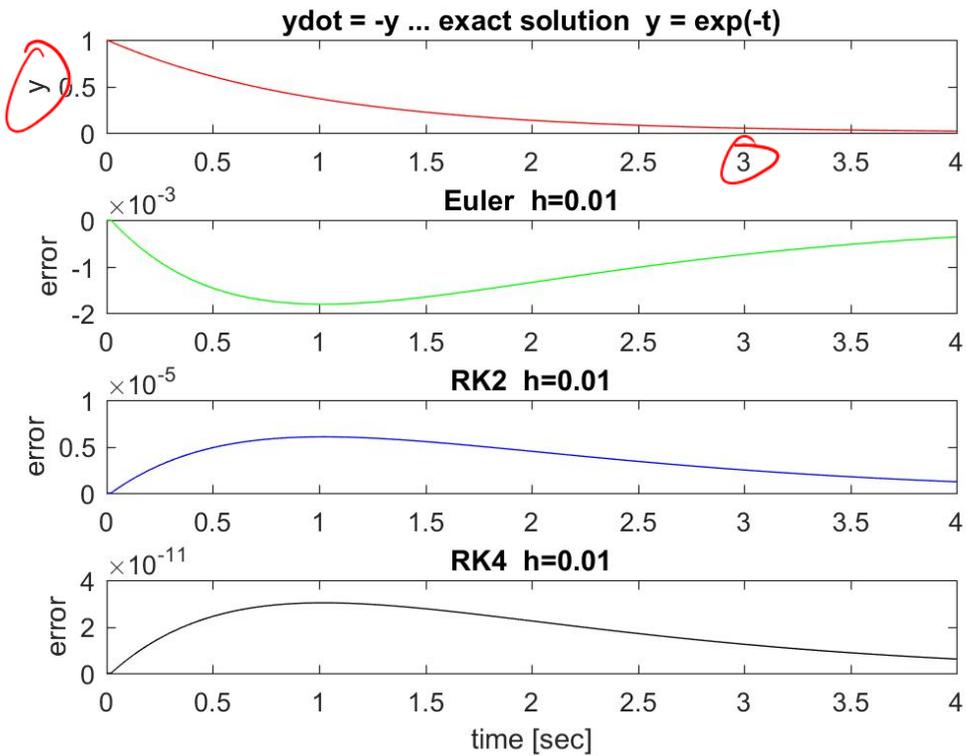
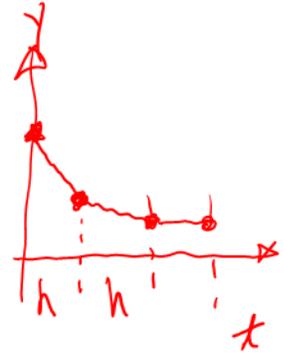
FIXED TIME STEP

- 1) Predictor and corrector functions are constant coefficient, weighted sums of f_i .
- 2) Coefficients are the same for all functions $\{\dot{y}\} = f(\{y\}, t)$.

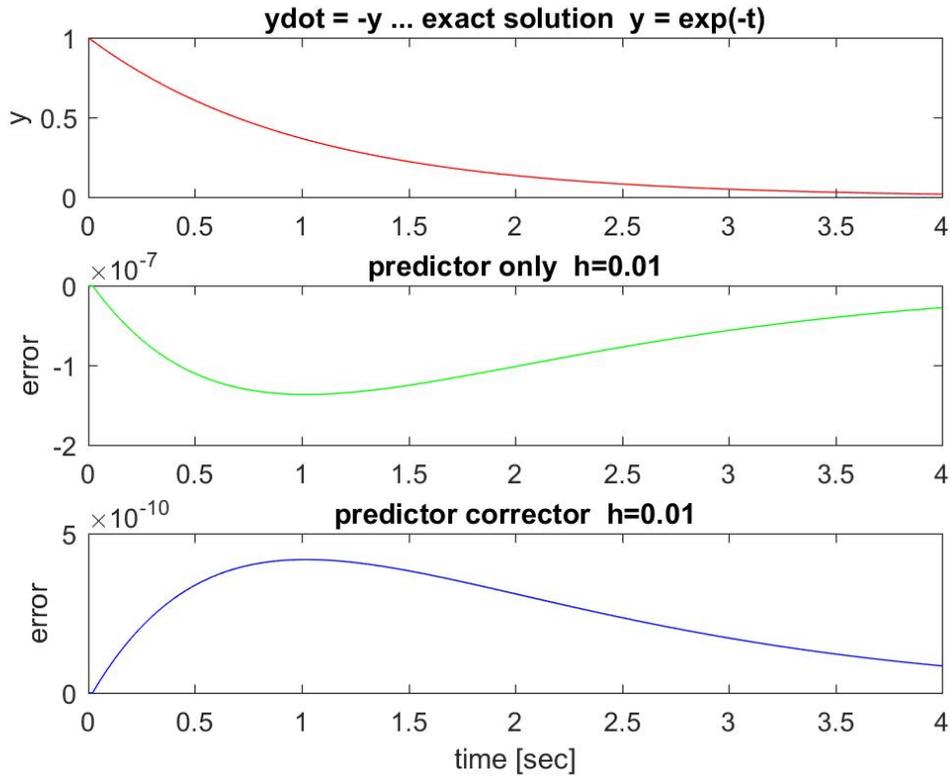
VARIABLE TIME STEP

- 1) Matrices $[A]$ and $[B]$ must be inverted at each time step.
- 2) Matrices $[A]$ and $[B]$ are the same for all functions $\{\dot{y}\} = f(\{y\}, t)$.

$\dot{y} = -y$



$\lambda = 10^{-2}$



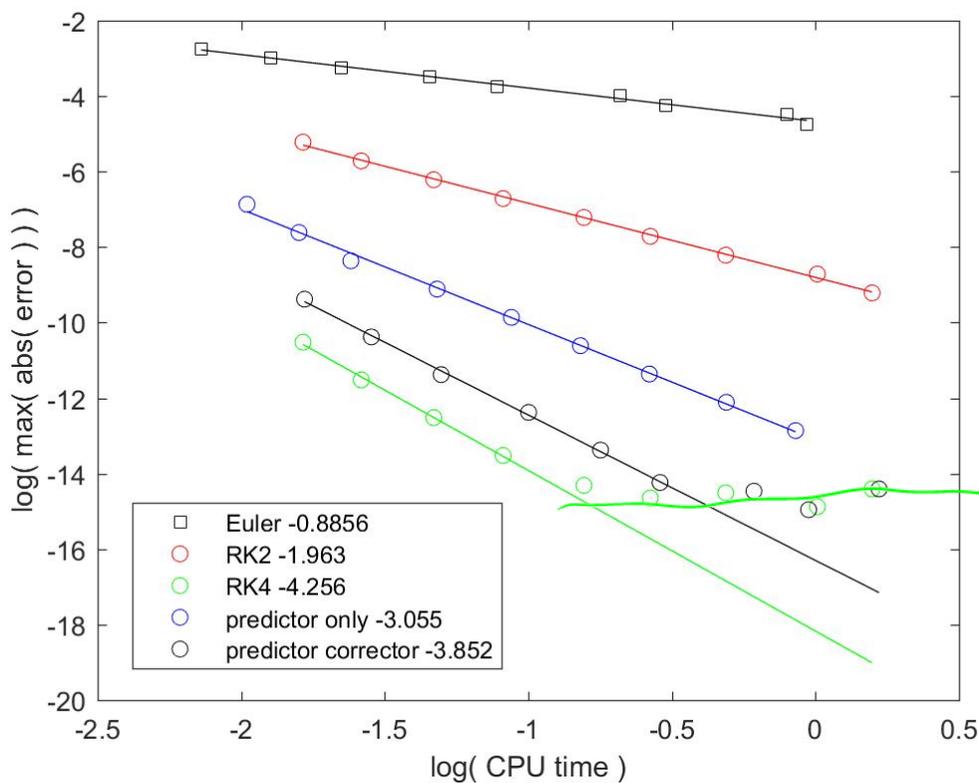
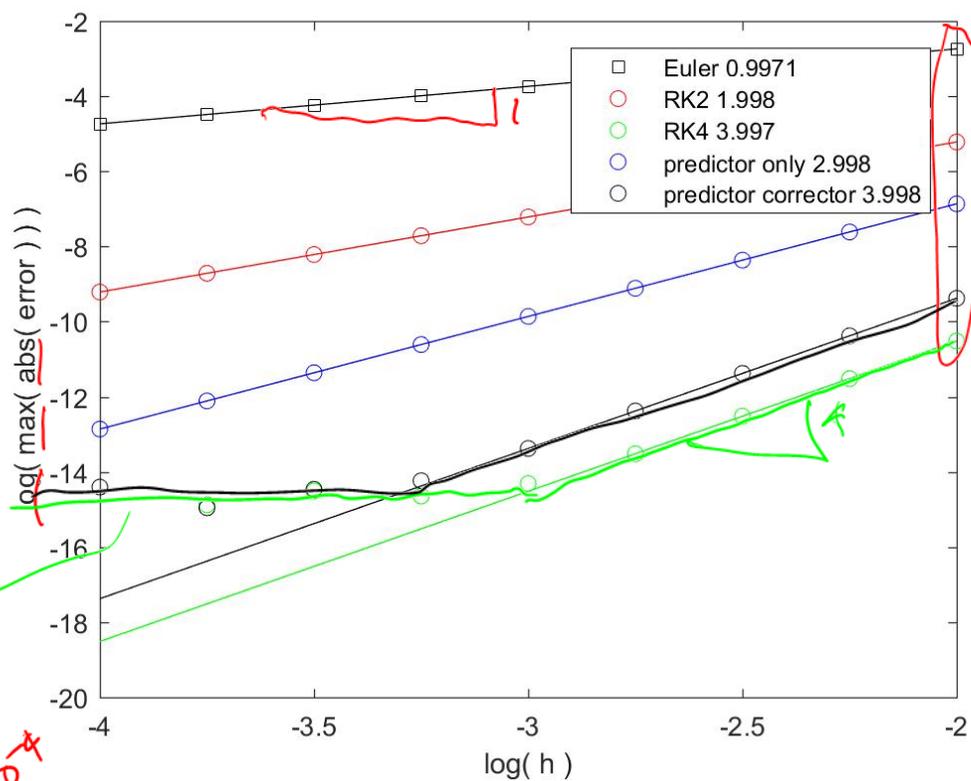
same speed as Euler

same speed as RK2

machine precision

$K=10^4$

$n=10^2$



```

% comp_int.m - compare integrators
%   time integration using Euler, RK2, RK4, predictor, corrector
%   simulate ydot = -y      exact solution y = exp(-t)
% HJSIII, 21.03.12

clear

% predictor-corrector coefficients
pred_coeff = [ 23  -16   5   ] / 12;
corr_coeff = [  9   19  -5   1 ] / 24;

% time span [sec]
tend = 4;

% time step [sec]
keep_err = [];
keep_cpu = [];
for log10_h = -4 : 0.25 : -2,

% time values
h = 10 ^ log10_h;
n = floor( tend / h ) + 1;
nm1 = n - 1;
t = h * (0:nm1)';

% exact solution
y_exact = exp( -t );

% allocate space for solutions
y_euler      = zeros(n,1);
y_rk2        = zeros(n,1);
y_rk4        = zeros(n,1);
y_pred_only  = zeros(n,1);
f_pred_only  = zeros(n,1);
y_pred_corr  = zeros(n,1);
f_pred_corr  = zeros(n,1);

% use exact solution for first three samples to get started
y_euler(1:3) = y_exact(1:3);
y_rk2(1:3)   = y_exact(1:3);
y_rk4(1:3)   = y_exact(1:3);
y_pred_only(1:3) = y_exact(1:3);
f_pred_only(1:3) = -y_pred_only(1:3);
y_pred_corr(1:3) = y_exact(1:3);
f_pred_corr(1:3) = -y_pred_corr(1:3);

% integrate forward in time

% Euler
tic;
for i = 3 : nm1,
    k = -y_euler(i);
    q=inv( rand(18,18) );           % function evaluation
    y_euler(i+1) = y_euler(i) + k * h;
end
cpu_euler = toc;

% RK2
tic;
for i = 3 : nm1,
    kA = -y_rk2(i);
    q=inv( rand(18,18) );           % function evaluation
    yB = y_rk2(i) + kA * h;
    kB = -yB;
    q=inv( rand(18,18) );           % function evaluation
    kM = (kA + kB) / 2;

```

```

    y_rk2(i+1) = y_rk2(i) + kM * h;
end
cpu_rk2 = toc;

% RK4
tic;
for i = 3 : nml,
    k1 = -y_rk4(i);
    q=inv( rand(18,18) );           % function evaluation
    y2 = y_rk4(i) + k1 * h/2;
    k2 = -y2;
    q=inv( rand(18,18) );           % function evaluation
    y3 = y_rk4(i) + k2 * h/2;
    k3 = -y3;
    q=inv( rand(18,18) );           % function evaluation
    y4 = y_rk4(i) + k3 * h;
    k4 = -y4;
    q=inv( rand(18,18) );           % function evaluation
    kM = (k1 + 2*k2 + 2*k3 + k4) / 6;
    y_rk4(i+1) = y_rk4(i) + kM * h;
end
cpu_rk4 = toc;

% predictor only
tic;
for i = 3 : nml,
    y_pred_only(i+1) = y_pred_only(i) ...
        + h * pred_coeff * [ f_pred_only(i)  f_pred_only(i-1)  f_pred_only(i-2) ]';
    f_pred_only(i+1) = -y_pred_only(i+1);
    q=inv( rand(18,18) );           % function evaluation
end
cpu_pred_only = toc;

% predictor-corrector
tic;
for i = 3 : nml,
    y_star = y_pred_corr(i) ...
        + h * pred_coeff * [ f_pred_corr(i)  f_pred_corr(i-1)  f_pred_corr(i-2) ]';
    f_star = -y_star;
    q=inv( rand(18,18) );           % function evaluation
    y_pred_corr(i+1) = y_pred_corr(i) ...
        + h * corr_coeff * [ f_star  f_pred_corr(i)  f_pred_corr(i-1)  f_pred_corr(i-2) ]';
    f_pred_corr(i+1) = -y_pred_corr(i+1);
    q=inv( rand(18,18) );           % function evaluation
end
cpu_pred_corr = toc;

% errors
err_euler    = y_euler - y_exact;
err_rk2      = y_rk2 - y_exact;
err_rk4      = y_rk4 - y_exact;
err_pred_only = y_pred_only - y_exact;
err_pred_corr = y_pred_corr - y_exact;

% plot exact solution and errors
figure( 1 )
clf
subplot( 4, 1, 1 )
plot( t,y_exact,'r' )
ylabel( 'y' )
title( 'ydot = -y ... exact solution y = exp(-t)' )
subplot( 4, 1, 2 )
plot( t,err_euler,'g' )
ylabel( 'error' )
title( [ 'Euler h=' num2str(h,4) ] )
subplot( 4, 1, 3 )
plot( t,err_rk2,'b' )
ylabel( 'error' )
title( [ 'RK2 h=' num2str(h,4) ] )
subplot( 4, 1, 4 )

```

```

plot( t,err_rk4,'k')
ylabel( 'error' )
title( [ 'RK4 h=' num2str(h,4) ] )
xlabel( 'time [sec]' )

figure( 2 )
clf
subplot( 3, 1, 1 )
plot( t,y_exact,'r' )
ylabel( 'y' )
title( 'ydot = -y ... exact solution y = exp(-t)' )
subplot( 3, 1, 2 )
plot( t,err_pred_only,'g')
ylabel( 'error' )
title( [ 'predictor only h=' num2str(h,4) ] )
subplot( 3, 1, 3 )
plot( t,err_pred_corr,'b' )
xlabel( 'time [sec]' )
ylabel( 'error' )
title( [ 'predictor corrector h=' num2str(h,4) ] )

keep_err = [ keep_err ; [ log10_h log10(max(abs(err_euler))) ...
                        log10(max(abs(err_rk2)))          log10(max(abs(err_rk4)))
...
                        log10(max(abs(err_pred_only)))
log10(max(abs(err_pred_corr))) ] ];

keep_cpu = [ keep_cpu ; [ log10_h log10(cpu_euler)          log10(cpu_rk2)          log10(cpu_rk2)
...
                        log10(cpu_pred_only) log10(cpu_pred_corr) ] ];

end % bottom - for log10_h

h = keep_err(:,1);
eeu = keep_err(:,2);
er2 = keep_err(:,3);
er4 = keep_err(:,4);
epo = keep_err(:,5);
epc = keep_err(:,6);

ceu = keep_cpu(:,2);
cr2 = keep_cpu(:,3);
cr4 = keep_cpu(:,4);
cpo = keep_cpu(:,5);
cpc = keep_cpu(:,6);

% slopes for error
poly_eeu = polyfit( h, eeu, 1 );
fit_eeu = polyval( poly_eeu, h );

poly_er2 = polyfit( h, er2, 1 );
fit_er2 = polyval( poly_er2, h );

poly_er4 = polyfit( h(6:end), er4(6:end), 1 );
fit_er4 = polyval( poly_er4, h );

poly_epo = polyfit( h, epo, 1 );
fit_epo = polyval( poly_epo, h );

poly_epc = polyfit( h(6:end), epc(6:end), 1 ); % ignore noise floor
fit_epc = polyval( poly_epc, h );

% plot error
figure( 3 )
clf
plot( h,eeu,'ks', h,er2,'ro', h,er4,'go', h,epo,'bo', h,epc,'ko', ...
      h,fit_eeu,'k', h,fit_er2,'r', h,fit_er4,'g', h,fit_epo,'b', h,fit_epc,'k' )
xlabel( 'log( h )' )
ylabel( 'log( max( abs( error ) ) )' )
legend( [ 'Euler ' num2str(poly_eeu(1),4) ], ...
        [ 'RK2 ' num2str(poly_er2(1),4) ], ...

```

```

    [ 'RK4 '          num2str(poly_er4(1),4) ], ...
    [ 'predictor only ' num2str(poly_epo(1),4) ], ...
    [ 'predictor corrector ' num2str(poly_epc(1),4) ] )

% slopes for CPU time
poly_ceu = polyfit( ceu, eeu, 1 );
fit_ceu = polyval( poly_ceu, ceu );

poly_cr2 = polyfit( cr2, er2, 1 );
fit_cr2 = polyval( poly_cr2, cr2 );

poly_cr4 = polyfit( cr4(6:end), er4(6:end), 1 );
fit_cr4 = polyval( poly_cr4, cr4 );

poly_cpo = polyfit( cpo, epo, 1 );
fit_cpo = polyval( poly_cpo, cpo );

poly_cpc = polyfit( cpc(6:end), epc(6:end), 1 ); % ignore noise floor
fit_cpc = polyval( poly_cpc, cpc );

% plot CPU time
figure( 4 )
clf
plot( ceu,eeu,'ks', cr2,er2,'ro', cr4,er4,'go', cpo,epo,'bo', cpc,epc,'ko', ...
      ceu,fit_ceu,'k', cr2,fit_cr2,'r', cr4,fit_cr4,'g', cpo,fit_cpo,'b', cpc,fit_cpc,'k' )
xlabel( 'log( CPU time )' )
ylabel( 'log( max( abs( error ) ) )' )
legend( [ 'Euler '          num2str(poly_ceu(1),4) ], ...
        [ 'RK2 '          num2str(poly_cr2(1),4) ], ...
        [ 'RK4 '          num2str(poly_cr4(1),4) ], ...
        [ 'predictor only ' num2str(poly_cpo(1),4) ], ...
        [ 'predictor corrector ' num2str(poly_cpc(1),4) ] )

% bottom of comp_int

```

Forward Time Integration in MATLAB

$[T, Y] = \text{solver}(\text{odefun}, \text{tspan}, y0)$
 $[t, Y] = \text{solver}(\text{odefun}, \text{tspan}, y0, \text{options})$

returns t, Y solution

odefun A function handle that evaluates the right side of the differential equations. All solvers solve systems of equations in the form $y' = f(t, y)$ or problems that involve a mass matrix, $M(t, y)y' = f(t, y)$. The `ode23s` solver can solve only equations with constant mass matrices. `ode15s` and `ode23t` can solve problems with a mass matrix that is singular, i.e., differential-algebraic equations (DAEs).

tspan A vector specifying the interval of integration, $[t0, tf]$. The solver imposes the initial conditions at `tspan(1)`, and integrates from `tspan(1)` to `tspan(end)`. To obtain solutions at specific times (all increasing or all decreasing), use `tspan = [t0, t1, ..., tf]`.

For `tspan` vectors with two elements $[t0 \ tf]$, the solver returns the solution evaluated at every integration step. For `tspan` vectors with more than two elements, the solver returns solutions evaluated at the given time points. The time values must be in order, either increasing or decreasing.

Specifying `tspan` with more than two elements does not affect the internal time steps that the solver uses to traverse the interval from `tspan(1)` to `tspan(end)`. All solvers in the ODE suite obtain output values by means of continuous extensions of the basic formulas. Although a solver does not necessarily step precisely to a time point specified in `tspan`, the solutions produced at the specified time points are of the same order of accuracy as the solutions computed at the internal time points.

Specifying `tspan` with more than two elements has little effect on the efficiency of computation, but for large systems, affects memory management.

y0 A vector of initial conditions.

options Structure of optional parameters that change the default integration properties. This is the fourth input argument.

*choose solver
provide function/subroutine
 $EY \rightarrow EY^0$
choose time span
provide EY at $t = t_{START}$*

output with variable time step

Solver	Problem Type	Order of Accuracy	When to Use
ode45	Nonstiff	Medium	Most of the time. This should be the first solver you try.
ode23	Nonstiff	Low	For problems with crude error tolerances or for solving moderately stiff problems.
ode113	Nonstiff	Low to high	For problems with stringent error tolerances or for solving computationally intensive problems.
ode15s	Stiff	Low to medium	If ode45 is slow because the problem is stiff.
ode23s	Stiff	Low	If using crude error tolerances to solve stiff systems and the mass matrix is constant.
ode23t	Moderately Stiff	Low	For moderately stiff problems if you need a solution without numerical damping.
ode23tb	Stiff	Low	If using crude error tolerances to solve stiff systems.

RK4

RK2

PECE

MATLAB Algorithms

`ode45` is based on an explicit Runge-Kutta (4,5) formula, the Dormand-Prince pair. It is a *one-step* solver – in computing $y(t_n)$, it needs only the solution at the immediately preceding time point, $y(t_{n-1})$. In general, `ode45` is the best function to apply as a *first try* for most problems. [3]

`ode23` is an implementation of an explicit Runge-Kutta (2,3) pair of Bogacki and Shampine. It may be more efficient than `ode45` at crude tolerances and in the presence of moderate stiffness. Like `ode45`, `ode23` is a one-step solver. [2]

`ode113` is a variable order Adams-Bashforth-Moulton PECE solver. It may be more efficient than `ode45` at stringent tolerances and when the ODE file function is particularly expensive to evaluate. `ode113` is a *multistep* solver — it normally needs the solutions at several preceding time points to compute the current solution. [7]

The above algorithms are intended to solve nonstiff systems. If they appear to be unduly slow, try using one of the stiff solvers below.

`ode15s` is a variable order solver based on the numerical differentiation formulas (NDFs). Optionally, it uses the backward differentiation formulas (BDFs, also known as Gear's method) that are usually less efficient. Like `ode113`, `ode15s` is a multistep solver. Try `ode15s` when `ode45` fails, or is very inefficient, and you suspect that the problem is stiff, or when solving a differential-algebraic problem. [9], [10]

`ode23s` is based on a modified Rosenbrock formula of order 2. Because it is a one-step solver, it may be more efficient than `ode15s` at crude tolerances. It can solve some kinds of stiff problems for which `ode15s` is not effective. [9]

`ode23t` is an implementation of the trapezoidal rule using a "free" interpolant. Use this solver if the problem is only moderately stiff and you need a solution without numerical damping. `ode23t` can solve DAEs. [10]

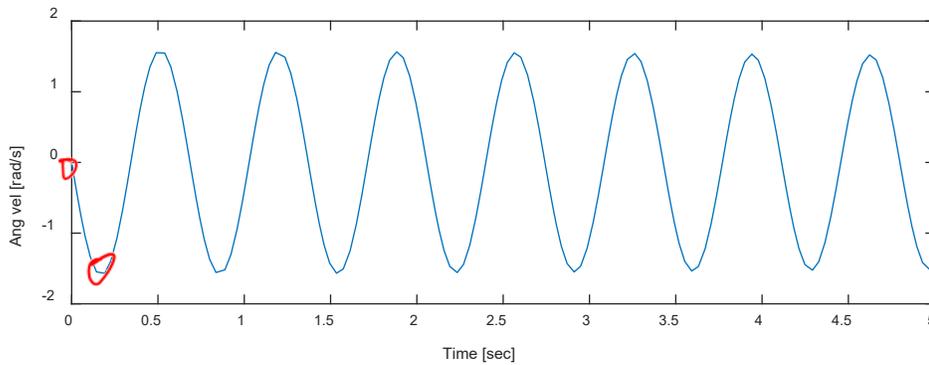
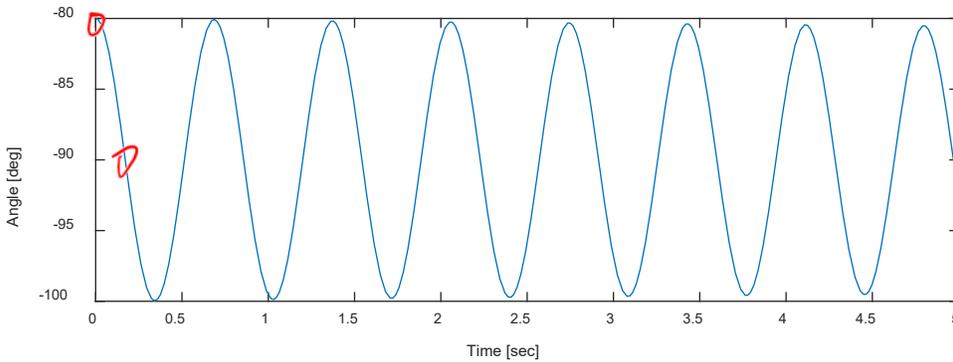
`ode23tb` is an implementation of TR-BDF2, an implicit Runge-Kutta formula with a first stage that is a trapezoidal rule step and a second stage that is a backward differentiation formula of order two. By construction, the same iteration matrix is used in evaluating both stages. Like `ode23s`, this solver may be more efficient than `ode15s` at crude tolerances. [8], [1]

References

- [1] Bank, R. E., W. C. Coughran, Jr., W. Fichtner, E. Grosse, D. Rose, and R. Smith, "Transient Simulation of Silicon Devices and Circuits," *IEEE Trans. CAD*, 4 (1985), pp. 436–451.
- [2] Bogacki, P. and L. F. Shampine, "A 3(2) pair of Runge-Kutta formulas," *Appl. Math. Letters*, Vol. 2, 1989, pp. 321–325.

- [3] Dormand, J. R. and P. J. Prince, "A family of embedded Runge-Kutta formulae," *J. Comp. Appl. Math.*, Vol. 6, 1980, pp. 19–26.
- [4] Forsythe, G. , M. Malcolm, and C. Moler, *Computer Methods for Mathematical Computations*, Prentice-Hall, New Jersey, 1977.
- [5] Kahaner, D. , C. Moler, and S. Nash, *Numerical Methods and Software*, Prentice-Hall, New Jersey, 1989.
- [6] Shampine, L. F. , *Numerical Solution of Ordinary Differential Equations*, Chapman & Hall, New York, 1994.
- [7] Shampine, L. F. and M. K. Gordon, *Computer Solution of Ordinary Differential Equations: the Initial Value Problem*, W. H. Freeman, SanFrancisco, 1975.
- [8] Shampine, L. F. and M. E. Hosea, "Analysis and Implementation of TR-BDF2," *Applied Numerical Mathematics* 20, 1996.
- [9] Shampine, L. F. and M. W. Reichelt, "The MATLAB ODE Suite," *SIAM Journal on Scientific Computing*, Vol. 18, 1997, pp. 1–22.
- [10] Shampine, L. F., M. W. Reichelt, and J.A. Kierzenka, "Solving Index-1 DAEs in MATLAB and Simulink," *SIAM Review*, Vol. 41, 1999, pp. 538–552.

Simple pendulum from Notes_09_02



```

ode_pendulum_main.m - main program for example use of ODE solver
simple_pendulum_notes_09_02
HJSIII, 20.04.09
%
% ODE coded in file ode_pendulum_yd.m - (JG+m*a*a)*thdd = T - m*g*a*cos(th)
%
{y} = { th }      {yd} = { thd }
      { thd }      { thdd }

clear

% global constants
global m JG a g

% constants
d2r = pi / 180;

% physical constants
% th [rad]
% thd [rad/sec]
% thdd [rad/sec^2]
m = 0.46;      % mass [lbm]
JG = 1.5;     % mass moment [lbm.in.in]
a = 3.7;     % centroid offset [in]
g = 386;     % acceleration of gravity [ipss]

% initial conditions
th0 = -80 * d2r; % 10 degrees above vertical
y0 = [ th0 0 ]'; % free release

% time range
tspan = [ 0 5 ];
    
```

ODE

$$\begin{Bmatrix} y \end{Bmatrix} = \begin{Bmatrix} \theta \\ \dot{\theta} \end{Bmatrix} \quad \begin{Bmatrix} y^0 \end{Bmatrix} = \begin{Bmatrix} \theta^0 \\ \dot{\theta}^0 \end{Bmatrix}$$

$$\begin{Bmatrix} y \end{Bmatrix}_{\text{INIT}} = \begin{Bmatrix} -20 \\ 0 \end{Bmatrix}$$

```

% ode23
[ t, y ] = ode23( 'ode_pendulum_yd', tspan, y0 );
th = y(:,1) / d2r; % [deg]
thd = y(:,2); % [rad/s]

% time domain results
figure( 1 )
clf
subplot( 2, 1, 1 )
plot( t, th )
xlabel( 'Time [sec]' )
ylabel( 'Angle [deg]' )
% axis( [ 0 5 -0.1 0.1 ] )

subplot( 2, 1, 2 )
plot( t, thd )
xlabel( 'Time [sec]' )
ylabel( 'Ang vel [rad/s]' )
% axis( [ 0 5 -1.5 1.5 ] )

% bottom - ode_pendulum_main

*****
function yd = ode_pendulum_yd( t, y )
% provides yd for integration
% simple pendulum Notes_09_02
% (JG+m*a*a)*thdd = T - m*g*a*cos(th)
% HJSIII, 20.04.09

% global constants
global m JG a g

% free motion
T = 0;

% individual terms
th = y(1);
thd = y(2);

% free swing
thdd = ( T - m*g*a*cos(th) ) / ( JG + m*a*a );

% simple Coulomb friction in revolute joint with bushing
mu = 0.1; % coefficient of friction
radius = 0.25; % [inch] radius for ID of bushing
W = m*g; % [lbm.in/s/s] weight of pendulum
Tf = mu * radius * W; % [lbf.in] frictional torque
%thdd = ( T -Tf*sign( thd ) -m*g*a*cos(th) ) / ( JG + m*a*a );

yd(1,1) = thd;
yd(2,1) = thdd;

return
% bottom - ode_pendulum_yd

```

use ode23 RK2

$$\{y\} = \begin{Bmatrix} \theta \\ \dot{\theta} \end{Bmatrix}$$

extract θ & $\dot{\theta}$

$y_d = \text{ode_pendulum_yd}(t, y)$

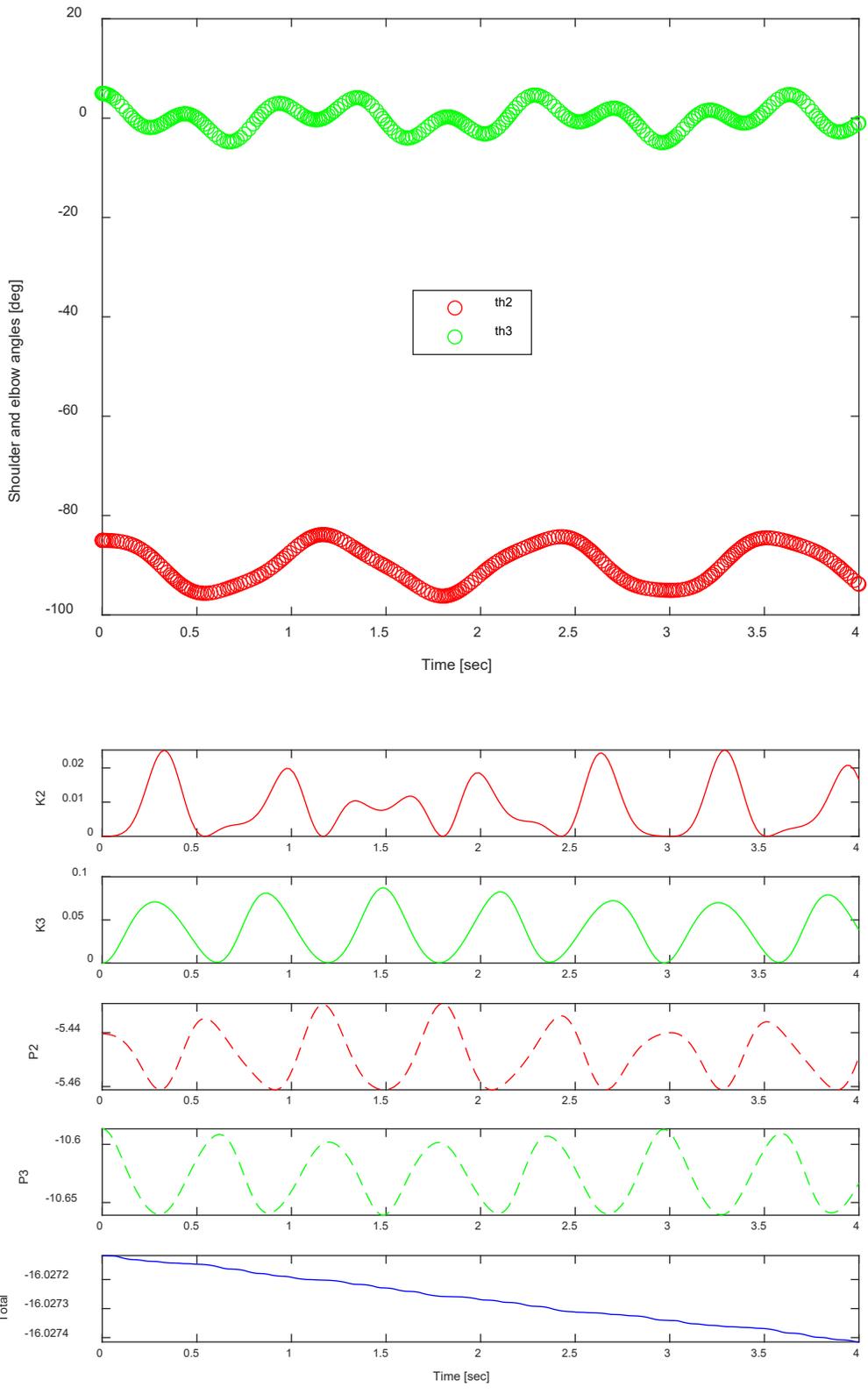
encodes in function subroutine

ODE

return

$$\{y\} = \begin{Bmatrix} \theta \\ \dot{\theta} \end{Bmatrix}$$

Double pendulum from Notes_09_02



```

% ode_dp_main.m - main program for example use of ODE solver
% double pendulum Notes_09_02
% HJSIII, 20.04.09
%
% ODE coded in file ode_dp_yd.m
%
% {y} = { theta2      }      {ydot} = { theta2_dot      }
%       { theta3      }      = ( theta3_dot      )
%       { theta2_dot  }      = ( theta2_dot_dot  )
%       { theta3_dot  }      = ( theta3_dot_dot  )

clear

% constants
d2r = pi /180;

% initial conditions - shoulder 5 deg above vertical, elbow 5 deg up
% th2=0 shoulder horizontal, th3=0 elbow straight
theta2_start = -85 * d2r;
theta3_start = 5 * d2r;

% free release
y0 = [ theta2_start  theta3_start  0  0  ]';

% time range
tspan = [ 0  4 ];

% time integration
[ t, y ] = ode23( 'ode_dp_yd', tspan, y0 );

% time domain results
th2 = y(:,1);
th3 = y(:,2);
th2d = y(:,3);
th3d = y(:,4);
figure( 1 )
  clf
  plot( t,th2/d2r,'ro', t,th3/d2r,'go' )
  xlabel( 'Time [sec]' )
  ylabel( 'Shoulder and elbow angles [deg]' )
  legend( 'th2', 'th3' )

% check kinetic and potential energy

% units = m kg sec m/sec m/sec/sec N N.m
gravity = 9.81; % [m/sec/sec]

% approximate human arm
d2 = 0.293; % [m] upper arm
d3 = 0.225; % [m] forearm

m2 = 3.80; % [kg] upper arm
m3 = 2.68; % [kg] forearm

J2 = 33300e-6; % [kg.m.m] upper arm
J3 = 9900e-6; % [kg.m.m] forearm

% symmetric links
a2 = d2 / 2;
a3 = d3 / 2;

% positions and velocities
x2 = a2*cos(th2);
y2 = a2*sin(th2);

x2d = -a2*th2d.*sin(th2);
y2d = a2*th2d.*cos(th2);

x3 = d2*cos(th2) + a3*cos(th2+th3);
y3 = d2*sin(th2) + a3*sin(th2+th3);

```

```

x3d = -d2*th2d.*sin(th2) - a3*(th2d+th3d).*sin(th2+th3);
y3d = d2*th2d.*cos(th2) + a3*(th2d+th3d).*cos(th2+th3);

% kinetic
K2 = m2*( x2d.*x2d + y2d.*y2d )/2 + J2*th2d.*th2d/2;
K3 = m3*( x3d.*x3d + y3d.*y3d )/2 + J3*(th2d+th3d).*(th2d+th3d)/2;

% potential
P2 = m2*y2*gravity;
P3 = m3*y3*gravity;

% show total energy
TE = K2 + K3 + P2 + P3;
figure( 2 )
clf
subplot( 5,1,1 )
plot( t,K2,'r-' )
ylabel( 'K2' )

subplot( 5,1,2 )
plot( t,K3,'g-' )
ylabel( 'K3' )

subplot( 5,1,3 )
plot( t,P2,'r--' )
ylabel( 'P2' )

subplot( 5,1,4 )
plot( t,P3,'g--' )
ylabel( 'P3' )

subplot( 5,1,5 )
plot( t,TE,'b' )
xlabel( 'Time [sec]' )
ylabel( 'Total' )

% bottom - ode_dp_main
*****
function ydot = ode_dp_yd( t, y )
% provides yd for integration
% double pendulum Notes_09_02
% HJSIII, 20.04.09

% units = m kg sec m/sec m/sec/sec N N.m
gravity = 9.81; % [m/sec/sec]

% approximate human arm
d2 = 0.293; % [m] upper arm
d3 = 0.225; % [m] forearm

m2 = 3.80; % [kg] upper arm
m3 = 2.68; % [kg] forearm

J2 = 33300e-6; % [kg.m.m] upper arm
J3 = 9900e-6; % [kg.m.m] forearm

% symmetric links
a2 = d2 / 2;
a3 = d3 / 2;

% extract state variables
th2 = y(1);
th3 = y(2);
th2d = y(3);
th3d = y(4);

% coefficients
JB = m3*a3*a3 + J3;
JA = JB + m2*a2*a2 + m3*d2*d2 + J2 + 2*m3*d2*a3*cos(th3);

```

```
C = JB + m3*d2*a3*cos(th3);
D = m3*d2*a3*sin(th3);

G2 = (m2*a2+m3*d2) * gravity * cos(th2);
G3 = m3*a3 * gravity * cos(th2+th3);

% motor torques
T2 = 0;
T3 = 0;

% test
M = [ JA  C  ;
      C  JB ];
rhs = [ T2+D*th3d*th3d+2*D*th2d*th3d-G2-G3 ;
        T3-D*th2d*th2d-G3 ];
thdd = inv(M) * rhs;

% return derivatives of state derivatives
ydot = [ th2d  th3d  thdd(1)  thdd(2) ]';

return
% bottom - ode_dp_yd
```

Second-Order Initial-Value Problems

Taylor Series Using Second Derivatives

given second-order initial-value differential equations of the form

$$\{\dot{y}\} = f(\{y\}, t) \text{ and } \{\ddot{y}\} = g(\{y\}, \{\dot{y}\}, t)$$

must know y_i at time t_i

may determine $\dot{y}_i = f_i$ and $\ddot{y}_i = g_i$

for time step $h = t_{i+1} - t_i$

$$y_{i+1} = y_i + f_i h + \frac{1}{2} g_i h^2$$

Fourth Order Runge-Kutta-Nystrom Using Second Derivatives (Fehlberg)

given second-order initial-value differential equations of the form

$$\{\dot{y}\} = f(\{y\}, t) \text{ and } \{\ddot{y}\} = g(\{y\}, \{\dot{y}\}, t)$$

must know y_i at time t_i

may determine $\dot{y}_i = f_i$ and $\ddot{y}_i = g_i$

$$k_1 = g(y_i, t_i)$$

$$k_2 = g\left(y_i + \frac{1}{3} h f_i + \frac{1}{18} h^2 k_1, t_i + \frac{1}{3} h\right)$$

$$k_3 = g\left(y_i + \frac{2}{3} h f_i + \frac{2}{9} h^2 k_2, t_i + \frac{2}{3} h\right)$$

$$k_4 = g\left(y_i + h f_i + \frac{1}{6} h^2 (2k_1 + k_3), t_i + h\right)$$

$$y_{i+1} = y_i + h f_i + \frac{1}{120} h^2 (13 k_1 + 36 k_2 + 9 k_3 + 2 k_4)$$

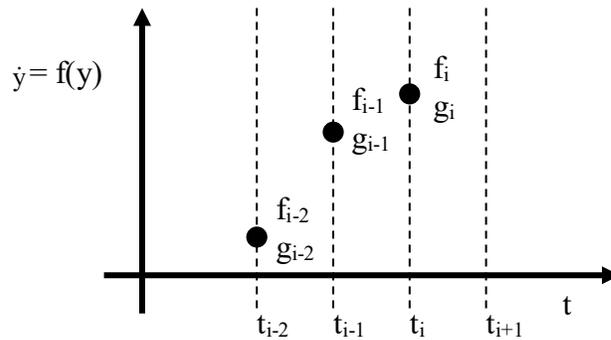
Fifth-degree (Quintic) Predictor and Seventh-degree (Heptic) Corrector with Second Derivatives

given second-order initial-value differential equations of the form

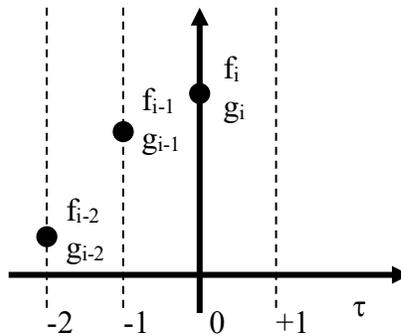
$$\{\dot{y}\} = f(\{y\}, t) \text{ and } \{\ddot{y}\} = g(\{y\}, \{\dot{y}\}, t)$$

must know $\{y\}_i, \{y\}_{i-1}, \{y\}_{i-2}$ respectively at times t_i, t_{i-1} and t_{i-2}

may determine $\{\dot{y}\}_i, \{\dot{y}\}_{i-1}, \{\dot{y}\}_{i-2}$ and $\{\ddot{y}\}_i, \{\ddot{y}\}_{i-1}, \{\ddot{y}\}_{i-2}$



fixed time step h $\tau = (t - t_i) / h$ $d\tau = dt / h$ $dt = h d\tau$ $d\tau / dt = 1 / h$



use fifth degree polynomial predictor $f = a_0 + a_1\tau + a_2\tau^2 + a_3\tau^3 + a_4\tau^4 + a_5\tau^5$

$$g = \frac{df}{dt} = \frac{\partial f}{\partial \tau} \frac{d\tau}{dt} \qquad g h = \frac{\partial f}{\partial \tau} \qquad g h = a_1 + 2a_2\tau + 3a_3\tau^2 + 4a_4\tau^3 + 5a_5\tau^4$$

$$\begin{Bmatrix} f \\ g \ h \end{Bmatrix} = \begin{bmatrix} 1 & \tau & \tau^2 & \tau^3 & \tau^4 & \tau^5 \\ 0 & 1 & 2\tau & 3\tau^2 & 4\tau^3 & 5\tau^4 \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{Bmatrix}$$

$$\begin{Bmatrix} f_i \\ f_{i-1} \\ f_{i-2} \\ g_i \ h \\ g_{i-1} \ h \\ g_{i-2} \ h \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & -2 & 4 & -8 & 16 & -32 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 3 & -4 & 5 \\ 0 & 1 & -4 & 12 & -32 & 80 \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{Bmatrix}$$

$$\begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{Bmatrix} = \begin{bmatrix} 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 \\ -23 & 16 & 7 & 12 & 16 & 2 \\ -33 & 16 & 17 & 13 & 32 & 5 \\ -17 & 4 & 13 & 6 & 20 & 4 \\ -3 & 0 & 3 & 1 & 4 & 1 \end{bmatrix} \begin{Bmatrix} f_i \\ f_{i-1} \\ f_{i-2} \\ g_i \ h \\ g_{i-1} \ h \\ g_{i-2} \ h \end{Bmatrix} / 4$$

know coefficients a_i for interpolant $f(\tau) = a_0 + a_1\tau + a_2\tau^2 + a_3\tau^3 + a_4\tau^4 + a_5\tau^5 = \dot{y}(\tau)$

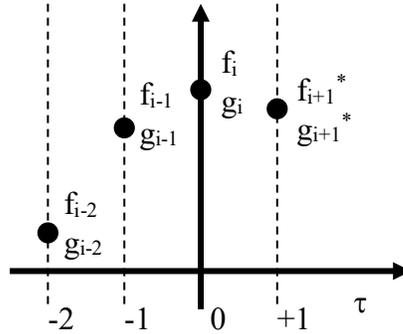
$$y_{i+1} = \int_{t_i}^{t_{i+1}} \dot{y} \ dt + y_i = h \int_0^1 f \ d\tau + y_i = y_i + h(a_0\tau + a_1\tau^2/2 + a_2\tau^3/3 + a_3\tau^4/4 + a_4\tau^5/5 + a_5\tau^6/6) \Big|_0^1$$

$$y_{i+1} = y_i + h(a_0 + a_1/2 + a_2/3 + a_3/4 + a_4/5 + a_5/6)$$

PREDICTOR (3-step Obreshkov)

$$y_{i+1}^* = y_i + h(-949 f_i + 608 f_{i-1} + 581 f_{i-2} + h(637 g_i + 1080 g_{i-1} + 173 g_{i-2}))/240$$

now use predicted value of y_{i+1}^* to compute $f(y_{i+1}^*) = f_{i+1}^*$ and $g(y_{i+1}^*) = g_{i+1}^*$



use seventh degree polynomial corrector

$$f(\tau) = b_0 + b_1\tau + b_2\tau^2 + b_3\tau^3 + b_4\tau^4 + b_5\tau^5 + b_6\tau^6 + b_7\tau^7 = \dot{y}(\tau)$$

$$\begin{Bmatrix} f \\ g \end{Bmatrix} = \begin{bmatrix} 1 & \tau & \tau^2 & \tau^3 & \tau^4 & \tau^5 & \tau^6 & \tau^7 \\ 0 & 1 & 2\tau & 3\tau^2 & 4\tau^3 & 5\tau^4 & 6\tau^5 & 7\tau^6 \end{bmatrix} \begin{Bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{Bmatrix}$$

$$\begin{Bmatrix} f_{i+1}^* \\ f_i \\ f_{i-1} \\ f_{i-2} \\ g_{i+1}^* \\ g_i \\ g_{i-1} \\ g_{i-2} \end{Bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & -2 & 4 & -8 & 16 & -32 & 64 & -128 \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 3 & -4 & 5 & -6 & 7 \\ 0 & 1 & -4 & 12 & -32 & 80 & -192 & 448 \end{bmatrix} \begin{Bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{Bmatrix}$$

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} = \begin{bmatrix} 0 & 108 & 0 & 0 & 0 & 0 & 0 & 01 \\ 0 & 0 & 0 & 0 & 0 & 108 & 0 & 0 \\ 56 & -297 & 216 & 25 & -12 & 108 & 108 & 6 \\ 124 & -27 & -108 & 11 & -24 & -189 & 0 & 3 \\ 50 & 270 & -270 & -50 & -3 & -216 & -189 & -12 \\ -59 & 54 & 27 & -22 & 21 & 54 & -27 & -6 \\ -52 & -81 & 108 & 25 & 15 & 108 & 81 & 6 \\ -11 & -27 & 27 & 11 & 3 & 27 & 27 & 3 \end{bmatrix} \begin{pmatrix} f_{i+1}^* \\ f_i \\ f_{i-1} \\ f_{i-2} \\ g_{i+1}^* \\ g_i \\ g_{i-1} \\ g_{i-2} \end{pmatrix} / 108$$

know coefficients b_i $f(\tau) = b_0 + b_1\tau + b_2\tau^2 + b_3\tau^3 + b_4\tau^4 + b_5\tau^5 + b_6\tau^6 + b_7\tau^7 = \dot{y}(\tau)$

$$\begin{aligned} y_{i+1} &= \int_{t_i}^{t_{i+1}} \dot{y} dt + y_i = h \int_0^1 f d\tau + y_i \\ &= y_i + h(b_0\tau + b_1\tau^2/2 + b_2\tau^3/3 + b_3\tau^4/4 + b_4\tau^5/5 + b_5\tau^6/6 + b_6\tau^7/7 + b_7\tau^8/8) \Big|_0^1 \end{aligned}$$

$$y_{i+1} = y_i + h(b_0 + b_1/2 + b_2/3 + b_3/4 + b_4/5 + b_5/6 + b_6/7 + b_7/8)$$

CORRECTOR (4-step Obreshkov)

$$y_{i+1} = y_i + h \left(\begin{aligned} &34465 f_{i+1}^* + 42255 f_i + 12015 f_{i-1} + 1985 f_{i-2} \\ &+ h(-3849 g_{i+1}^* + 22977 g_i + 7263 g_{i-1} + 489 g_{i-2}) \end{aligned} \right) / 90720$$

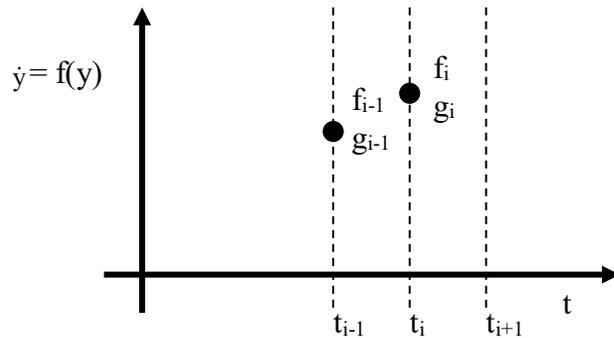
Third-degree (Cubic) Predictor and Third-degree (Cubic) Corrector with Second Derivatives

given second-order initial-value differential equations of the form

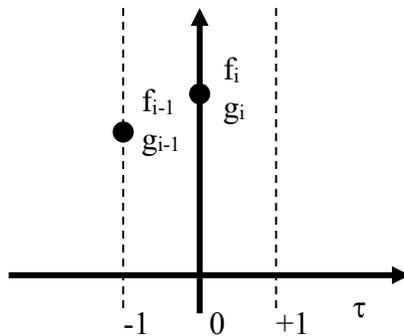
$$\dot{y} = f(y, t) \text{ and } \ddot{y} = g(y, \dot{y}, t)$$

must know $\{y\}_i$ and $\{y\}_{i-1}$ respectively at times t_i and t_{i-1}

may determine $\{\dot{y}\}_i$, $\{\dot{y}\}_{i-1}$ and $\{\ddot{y}\}_i$, $\{\ddot{y}\}_{i-1}$



fixed time step h $\tau = (t - t_i) / h$ $d\tau = dt / h$ $dt = h d\tau$ $d\tau / dt = 1 / h$



use third degree polynomial predictor $f = a_0 + a_1\tau + a_2\tau^2 + a_3\tau^3$

$$g = \frac{df}{dt} = \frac{\partial f}{\partial \tau} \frac{d\tau}{dt} \quad g h = \frac{\partial f}{\partial \tau} \quad g h = a_1 + 2a_2\tau + 3a_3\tau^2$$

$$\begin{Bmatrix} f \\ g h \end{Bmatrix} = \begin{bmatrix} 1 & \tau & \tau^2 & \tau^3 \\ 0 & 1 & 2\tau & 3\tau^2 \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{Bmatrix}$$

$$\begin{Bmatrix} f_i \\ f_{i-1} \\ g_i h \\ g_{i-1} h \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & -1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & -2 & 3 \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{Bmatrix}$$

$$\begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & 2 & 1 \\ -2 & 2 & 1 & 1 \end{bmatrix} \begin{Bmatrix} f_i \\ f_{i-1} \\ g_i h \\ g_{i-1} h \end{Bmatrix}$$

know coefficients a_i for interpolant $f(\tau) = a_0 + a_1\tau + a_2\tau^2 + a_3\tau^3 = \dot{y}(\tau)$

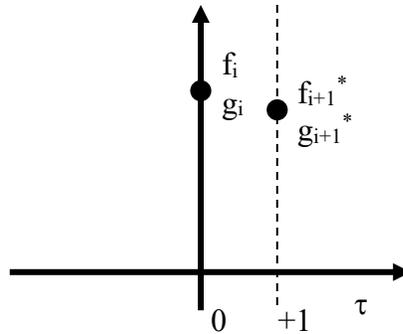
$$y_{i+1} = \int_{t_i}^{t_{i+1}} \dot{y} dt + y_i = h \int_0^1 f d\tau + y_i = y_i + h \left(a_0\tau + a_1\tau^2/2 + a_2\tau^3/3 + a_3\tau^4/4 \right)_0^1$$

$$y_{i+1} = y_i + h(a_0 + a_1/2 + a_2/3 + a_3/4)$$

PREDICTOR (2-step Obreshkov per Sehnalova)

$$y_{i+1}^* = y_i + h(-6f_i + 18f_{i-1} + h(17g_i + 7g_{i-1}))/12$$

now use predicted value of y_{i+1}^* to compute $f(y_{i+1}^*) = f_{i+1}^*$ and $g(y_{i+1}^*) = g_{i+1}^*$



use third degree polynomial corrector $f(\tau) = b_0 + b_1\tau + b_2\tau^2 + b_3\tau^3 = \dot{y}(\tau)$

$$\begin{Bmatrix} f \\ g h \end{Bmatrix} = \begin{bmatrix} 1 & \tau & \tau^2 & \tau^3 \\ 0 & 1 & 2\tau & 3\tau^2 \end{bmatrix} \begin{Bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{Bmatrix}$$

$$\begin{Bmatrix} f_{i+1} \\ f_i \\ g_{i+1} h \\ g_i h \end{Bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{Bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{Bmatrix}$$

$$\begin{Bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{Bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 3 & -3 & -1 & -2 \\ -2 & 2 & 1 & 1 \end{bmatrix} \begin{Bmatrix} f_{i+1} \\ f_i \\ g_{i+1} h \\ g_i h \end{Bmatrix}$$

know coefficients b_i $f(\tau) = b_0 + b_1\tau + b_2\tau^2 + b_3\tau^3 = \dot{y}(\tau)$

$$\begin{aligned} y_{i+1} &= \int_{t_i}^{t_{i+1}} \dot{y} dt + y_i = h \int_0^1 f d\tau + y_i \\ &= y_i + h(b_0\tau + b_1\tau^2/2 + b_2\tau^3/3 + b_3\tau^4/4) \Big|_0^1 \end{aligned}$$

$$y_{i+1} = y_i + h(b_0 + b_1/2 + b_2/3 + b_3/4)$$

CORRECTOR (2-step Obreshkov per Sehnalova)

$$y_{i+1}^* = y_i + h(6f_{i+1} + 6f_i + h(-g_{i+1} + g_i))/12$$

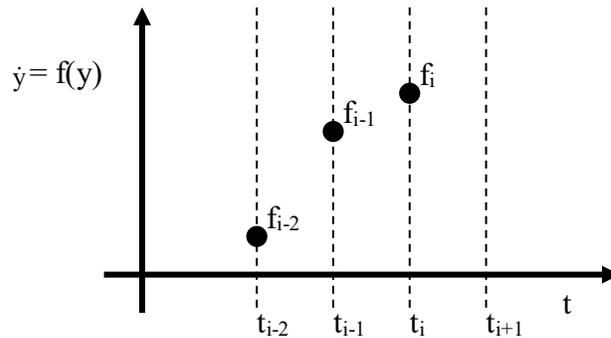
Least-Squares Quadratic Predictor and Cubic Corrector with Second Derivatives

given second-order initial-value differential equations of the form

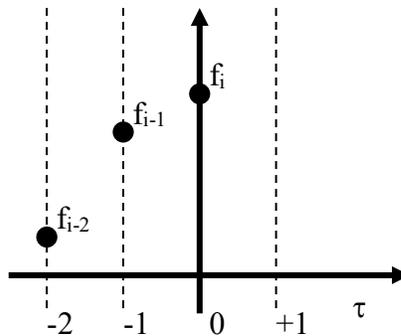
$$\dot{y} = f(y, t) \text{ and } \ddot{y} = g(y, \dot{y}, t)$$

must know $\{y\}_i, \{y\}_{i-1}, \{y\}_{i-2}$ respectively at times t_i, t_{i-1} and t_{i-2}

may determine $\{\dot{y}\}_i, \{\dot{y}\}_{i-1}, \{\dot{y}\}_{i-2}$ and $\{\ddot{y}\}_i, \{\ddot{y}\}_{i-1}, \{\ddot{y}\}_{i-2}$



fixed time step h $\tau = (t - t_i) / h$ $d\tau = dt / h$ $dt = h d\tau$ $d\tau / dt = 1 / h$



use quadratic polynomial predictor $f = a_0 + a_1 \tau + a_2 \tau^2$

$$g = \frac{df}{dt} = \frac{\partial f}{\partial \tau} \frac{d\tau}{dt} \quad g h = (a_1 + 2a_2 \tau) / h \quad g h = a_1 + 2a_2 \tau$$

$$\begin{Bmatrix} f \\ g h \end{Bmatrix} = \begin{bmatrix} 1 & \tau & \tau^2 \\ 0 & 1 & 2\tau \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \end{Bmatrix}$$

$$\begin{Bmatrix} f_i \\ g_i h \\ f_{i-1} \\ g_{i-1} h \\ f_{i-2} \\ g_{i-2} h \end{Bmatrix} \approx \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & -1 & 1 \\ 0 & 1 & -2 \\ 1 & -2 & 4 \\ 0 & 1 & -4 \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \end{Bmatrix}$$

$$\{Y\} \approx [X]\{\beta\} \quad \{Y\} = \begin{Bmatrix} f_i \\ g_i h \\ f_{i-1} \\ g_{i-1} h \\ f_{i-2} \\ g_{i-2} h \end{Bmatrix} \quad [X] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & -1 & 1 \\ 0 & 1 & -2 \\ 1 & -2 & 4 \\ 0 & 1 & -4 \end{bmatrix} \quad \{\beta\} = \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \end{Bmatrix}$$

$$\{\beta\} = ([X]^T [X])^{-1} [X]^T \{Y\} \quad \text{least squares solution}$$

$$\begin{Bmatrix} a_0 \\ a_1 \\ a_2 \end{Bmatrix} = \begin{bmatrix} 71 & 36 & 40 & 26 & 19 & 16 \\ 36 & 86 & -20 & 26 & -16 & -34 \\ 5 & 30 & -10 & 0 & 5 & -30 \end{bmatrix} \begin{Bmatrix} f_i \\ g_i h \\ f_{i-1} \\ g_{i-1} h \\ f_{i-2} \\ g_{i-2} h \end{Bmatrix} / 130$$

know coefficients a_0 , a_1 and a_2 for interpolant $f(\tau) = a_0 + a_1 \tau + a_2 \tau^2 = \dot{y}(\tau)$

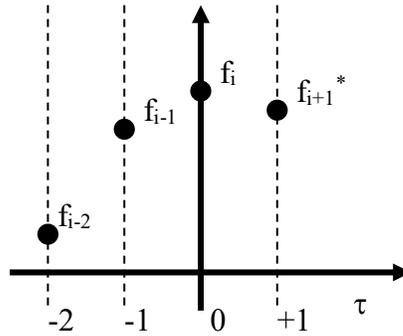
$$y_{i+1} = \int_{t_i}^{t_{i+1}} \dot{y} dt + y_i = h \int_0^1 f d\tau + y_i = h (a_0 \tau + a_1 \tau^2 / 2 + a_2 \tau^3 / 3) \Big|_0^1 + y_i$$

$$y_{i+1} = y_i + h (a_0 + a_1 / 2 + a_2 / 3)$$

PREDICTOR (least-squares 3-step Adams-Bashforth)

$$y_{i+1} = y_i + h (272 f_i + 80 f_{i-1} + 38 f_{i-2} + h (267 g_i + 117 g_{i-1} - 33 g_{i-2})) / 390$$

now use predicted value of y_{i+1} to compute $f(y_{i+1}) = f_{i+1}^*$ and $g(y_{i+1}) = g_{i+1}^*$



use cubic polynomial corrector $f = b_0 + b_1 \tau + b_2 \tau^2 + b_3 \tau^3$

$$\begin{Bmatrix} f \\ g h \end{Bmatrix} = \begin{bmatrix} 1 & \tau & \tau^2 & \tau^3 \\ 0 & 1 & 2\tau & 3\tau^2 \end{bmatrix} \begin{Bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{Bmatrix}$$

$$\begin{Bmatrix} f_{i+1}^* \\ g_{i+1}^* h \\ f_i \\ g_i h \\ f_{i-1} \\ g_{i-1} h \\ f_{i-2} \\ g_{i-1} h \end{Bmatrix} \approx \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & -1 & 1 & -1 \\ 0 & 1 & -2 & 3 \\ 1 & -2 & 4 & -8 \\ 0 & 1 & -4 & 12 \end{bmatrix} \begin{Bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{Bmatrix}$$

$$\begin{Bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{Bmatrix} = \begin{bmatrix} 754 & -438 & 851 & 175 & 542 & 374 & 241 & 159 \\ 590 & 126 & 175 & 593 & -374 & 394 & -391 & -471 \\ 32 & 618 & -203 & -67 & 4 & -266 & 167 & 21 \\ -45 & 213 & -69 & -111 & 69 & -111 & 45 & 213 \end{bmatrix} \begin{Bmatrix} f_{i+1}^* \\ g_{i+1}^* h \\ f_i \\ g_i h \\ f_{i-1} \\ g_{i-1} h \\ f_{i-2} \\ g_{i-1} h \end{Bmatrix} / 2388$$

know coefficients b_0, b_1, b_2 and b_3 for new interpolant $f(\tau) = b_0 + b_1 \tau + b_2 \tau^2 + b_3 \tau^3 = \dot{y}(\tau)$

$$y_{i+1} = \int_{t_i}^{t_{i+1}} \dot{y} dt + y_i = h \int_0^1 f d\tau + y_i = h (b_0\tau + b_1\tau^2/2 + b_2\tau^3/3 + b_3\tau^4/4) \Big|_0^1 + y_i$$

$$y_{i+1} = y_i + h (b_0 + b_1/2 + b_2/3 + b_3/4)$$

CORRECTOR (least-squares 4-step Adams-Moulton)

$$y_{i+1} = y_i + h \left(\begin{array}{l} 12581 f_{i+1}^* + 10243 f_i + 4483 f_{i-1} + 1349 f_{i-2} \\ + h (-1389 g_{i+1}^* + 5057 g_i + 5455 g_{i-1} - 195 g_{i-2}) \end{array} \right) / 28656$$

does not work as well as Obreshkov