Numerical Solutions to Differential Equations

Lecture Notes #14
Adaptive RKF45 Solver

Peter Blomgren,
⟨blomgren.peter@gmail.com⟩

Department of Mathematics and Statistics
Dynamical Systems Group
Computational Sciences Research Center
San Diego State University
San Diego, CA 92182-7720

http://terminus.sdsu.edu/

Spring 2015

Outline

Introduction

In the presence of stiffness we have seen that selecting one method with a single step-length may be very inefficient.

In the current lecture we engineer a method of leveraging the error estimate in the Runge-Kutta-Fehlberg-4-5 (RKF45) scheme in order to **automatically** decide what the appropriate time-step should be.

Starting Point: Modularized RK-Solver

Code: Fixed-step RKm-n Segment #1

```
function [tOut, yv, ev] = rk(f, y0, tRange, c, A, b1, b2)

%%(Make sure these are row vectors)
c  = reshape(c , 1, length(c));   y0 = reshape(y0, 1, length(y0));
b1 = reshape(b1, 1, length(b1));  b2 = reshape(b2, 1, length(b2));

%%(Coeffieients for error estimation)
E = (b2 - b1);

%%(Allocate space)
yv = zeros( length(tRange), length(y0) );  ev = yv;

%%(Insert starting value)
yv(1,:) =  y0;

%%(Get the size of A)
[rows,cols] = size(A);

%%(Give an error if A is not square)
assert(rows==cols, 'A matrix must be square')
```

Starting Point: Modularized RK-Solver

Code: Fixed-step RK*m*-*n* — Segment #2

```
%%(MAIN LOOP)
%%(Iterate over time range (index 'i' is "new time")
for i = 2 : length(tRange)
    h       = tRange(i) - tRange(i-1);
    k       = zeros( rows, length(y0) );
    yv(i,:) = yv(i-1,:);
    %%(Compute remaining k values)
    for j = 1:rows
      tk = tRange(i-1) + h*c(j);
      yk = yv(i-1,:);
      for n = 1:(j-1)
        yk = yk + h * A(j,n) * k(n,:);
      end
      k(j,:)  = f( tk, yk );
      yv(i,:) = yv(i,:) + h*b1(j)*k(j,:);
      ev(i,:) = ev(i,:) + h*E(j)*k(j,:);
    end
end

tOut = tRange;
```


What Has to Change???

- `tRange` will now contain the start and end times, only.
- Hence, we do not *a priori* know the size of `tOut`, `yv`, nor `ev`.
- We need to figure out how the step size `h` should change, in order to keep the step-error `steptol` (new variable) while keeping `h` within a prescribed range [`hmin`, `hmax`].
- Notice that in order to rescale `h` we need to know how the error of the "stepping" method scales, `steporder`.
- Our adaptive version will take one additional argument `opts` which is a matlab structure with the following entries:
  - `opts.h.min` — smallest allowable step
  - `opts.h.max` — largest allowable step
  - `opts.h.typical` — typical (initial step)
  - `opts.step.tol` — step tolerance
  - `opts.step.order` — stepping method order

Adaptive Runge Kutta

Code: Adaptive-step RK*m*-*n* — Segment#1

```
function [tOut, yv, ev] = rka(f, y0, tRange, c, A, b1, b2, opts)

%%(Make sure these are row vectors)
c  = reshape(c , 1, length(c));    y0 = reshape(y0, 1, length(y0));
b1 = reshape(b1, 1, length(b1));  b2 = reshape(b2, 1, length(b2));

%%(Coeffieients for error estimation)
E = (b2 - b1);

%%%(Allocate space)
yv = zeros( 1, length(y0) );  ev = zeros( 1, length(y0) );

%%(Insert starting value)
yv(1,:) =  y0;

%%(Get the size of A)
[rows,cols] = size(A);

%%(Give an error if A is not square)
assert(rows==cols, 'A matrix must be square')
```


Adaptive Runge Kutta

Code: Adaptive-step RK*m*-*n* — Segment#2

```
%%(Set up for main loop)
i    = 1;
t    = tRange(1);
tOut = tRange(1);
h    = opts.h.typical;
tol  = opts.step.tol;
```

Adaptive Runge Kutta

Code: Adaptive-step RKm-n Segment#3

```
%%(MAIN LOOP --- Iterate over time range)
while( t < tRange(2) )
    k          = zeros( rows, length(y0) );
    ytry       = yv(i,:);
    ev(i+1,:) = zeros(size(y0));
    %%(Compute remaining k values)
    for j = 1:rows
      tk = t + h*c(j);
      yk = yv(i,:);
      for n = 1:(j-1)
        yk = yk + h * A(j,n) * k(n,:);
      end
      k(j,:)     = f( tk, yk );
      ev(i+1,:) = ev(i+1,:) + h*E(j)*k(j,:);
      ytry       = ytry + h*b1(j)*k(j,:);
    end
```


Adaptive Runge Kutta

Code: Adaptive-step RKm-n Segment#4

```
    %%( Check if the error is small enough )
    errnorm = norm(ev(i+1,:));
    if( errnorm <= tol )
      i = i + 1;
      t = t + h;
      yv(i,:) = ytry;
      tOut(i) = t;
    end

    %%( Calculate Suggested Scaling Factor )
    ssf = ( (tol/2) / errnorm ) ^ (1/opts.step.order);

    %%( Limit scaling between 0.1 and 4 )
    ssf = min(max(0.1,ssf),4);
    h   = min( h * ssf, opts.h.max );
```


Adaptive Runge Kutta

Code: Adaptive-step RKm-n Segment#5

```
    %%( Warn user if the stepsize is too small )
    if( h < opts.h.min )
      warning(...
          sprintf(...
              'Stepsize %g smaller than threshold (%g) at time %g\n',...
              h, opts.h.min, t));
    end
    %%( Don't overstep the end of tRange )
    if( t+h > tRange(2) )
      h = tRange(2) - t;
    end
end
```


Scalar Problem

Code: Scalar Driver Segment #1

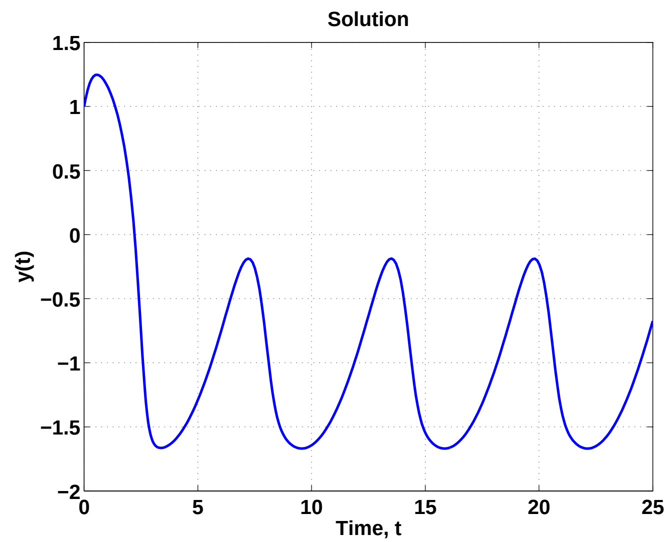
```
c   = [0 1/4 3/8 12/13 1 1/2];
A   = [0 0 0 0 0 0; 1/4 0 0 0 0 0; 3/32 9/32 0 0 0 0];
A   = [A; 1932/2197 -7200/2197 7296/2197 0 0 0];
A   = [A; 439/216 -8 3680/513 -845/4104 0 0];
A   = [A; -8/27 2 -3544/2565 1859/4104 -11/40 0];
b_1 = [ 25/216 0 1408/2565 2197/4104 -1/5 0];
b_2 = [ 16/135 0 6656/12825 28561/56430 -9/50 2/55];

f = @(t,y)(y - y^3 + 2*cos(t) - 1 );

t_start = 0;
t_final = 25;

opts.h.min      = eps^(2/3);
opts.h.max      = 0.1;
opts.h.typical  = 0.5;
opts.step.tol   = 10^(-6);
opts.step.order = 4;
```

Scalar Problem

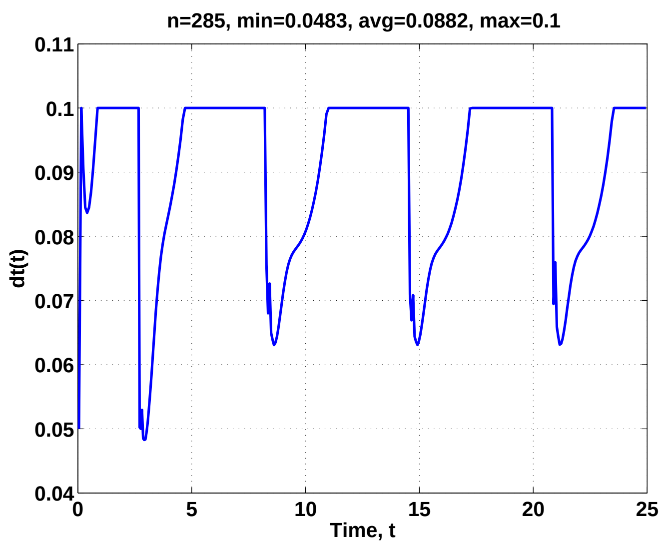
Code: Scalar Driver — Segment #2

```
[tv, yv, ev] = rka(f, 1, [t_start t_final], c, A, b_1, b_2, opts);

figure(1);  plot(tv, yv)
title('Solution');  ylabel('y(t)');  xlabel('Time, t');  grid on

n = length(tv)-1;
ts = diff(tv(1:n));
figure(2);  plot(tv(1:n),[NaN ts])
title(sprintf('n=%d, min=%.3g, avg=%.3g, max=%.3g',...
              n+1, min(ts), mean(ts), max(ts)))
ylabel('dt(t)');  xlabel('Time, t');  grid on

figure(3)
semilogy(tv(2:n),abs(ev(2:n)))
title('Estimated Step Error')
ylabel('err(t)');  xlabel('Time, t')
grid on
```

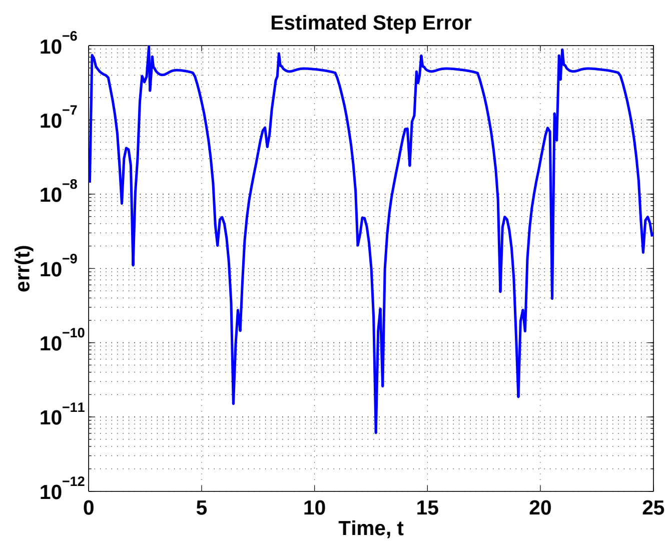

Results



Results



Results



Vector Valued Problem

Code: Vector Valued Driver Segment #1

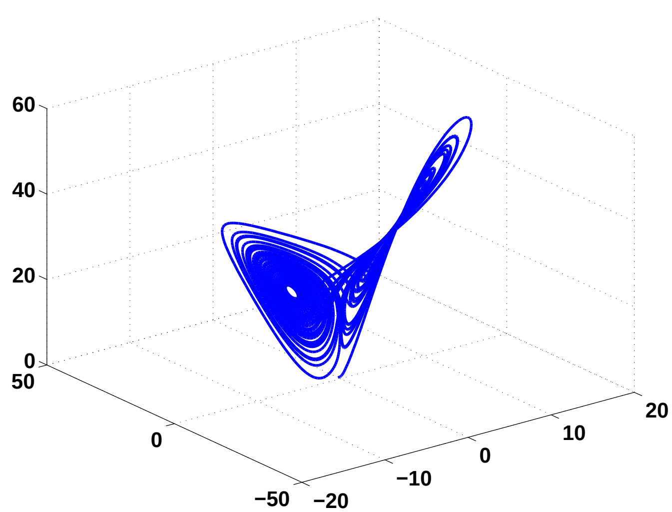
```
c   = [0 1/4 3/8 12/13 1 1/2];
A   = [0 0 0 0 0 0; 1/4 0 0 0 0 0; 3/32 9/32 0 0 0 0];
A   = [A; 1932/2197 -7200/2197 7296/2197 0 0 0];
A   = [A; 439/216 -8 3680/513 -845/4104 0 0];
A   = [A; -8/27 2 -3544/2565 1859/4104 -11/40 0];
b_1 = [ 25/216 0 1408/2565 2197/4104 -1/5 0];
b_2 = [ 16/135 0 6656/12825 28561/56430 -9/50 2/55];

r     = 28;
s     = 10;
b     = 8/3;

f = @(t,x)([s*(-x(1)+x(2));r*x(1)-x(2)-x(3)*x(1);-b*x(3)+x(1)*x(2)]);

opts.h.min      = eps^(2/3);
opts.h.max      = 0.1;
opts.h.typical  = 0.01;
opts.step.tol   = 10^(-6);
opts.step.order = 4;
```


Vector Valued Problem

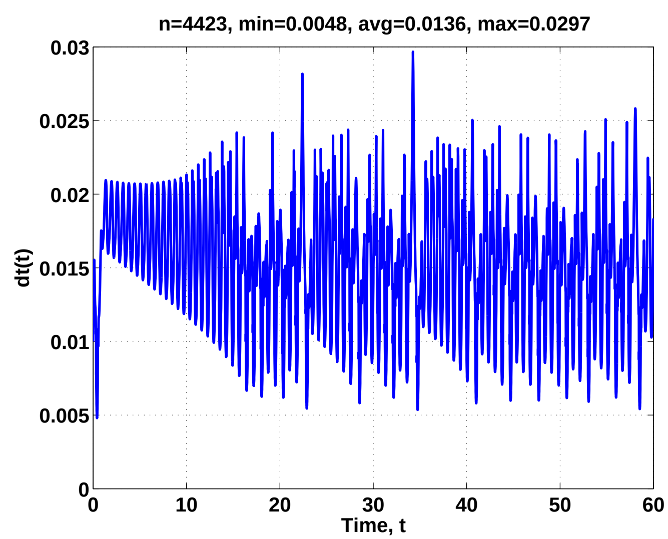
Code: Vector Valued Driver Segment #2

```
Tmax   = 60;
T      = [0 Tmax];
[tv,yv,ev] = rka(f, [0 1 0], T, c, A, b_1, b_2, opts);

figure(1); plot3(yv(:,1), yv(:,2), yv(:,3), 'k-'); grid on

n = length(tv)-1; ts = diff(tv(1:n));
figure(2);
plot(tv(1:n),[NaN ts])
title(sprintf('n=%d, min=%.3g, avg=%.3g, max=%.3g',...
              n+1, min(ts), mean(ts), max(ts)))
ylabel('dt(t)'); xlabel('Time, t'); grid on

figure(3)
rmserr = eps+sqrt(sum(abs(ev(2:n,:)).^2,2));
semilogy(tv(2:n),rmserr);
title('Estimated Step Error'); ylabel('err(t)'); xlabel('Time, t')
grid on
```

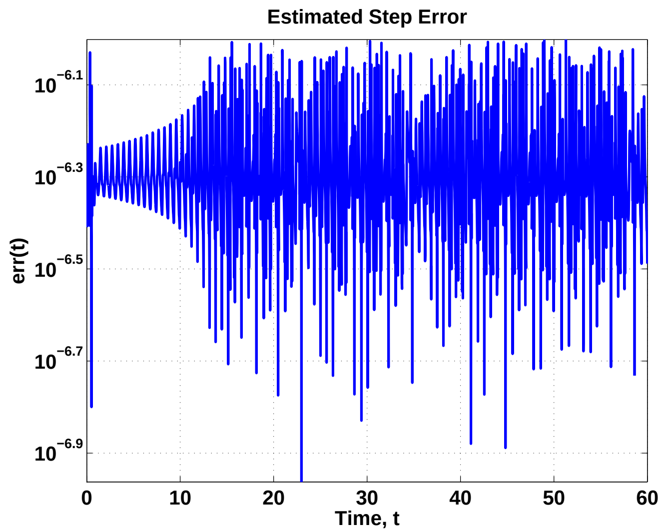

Results



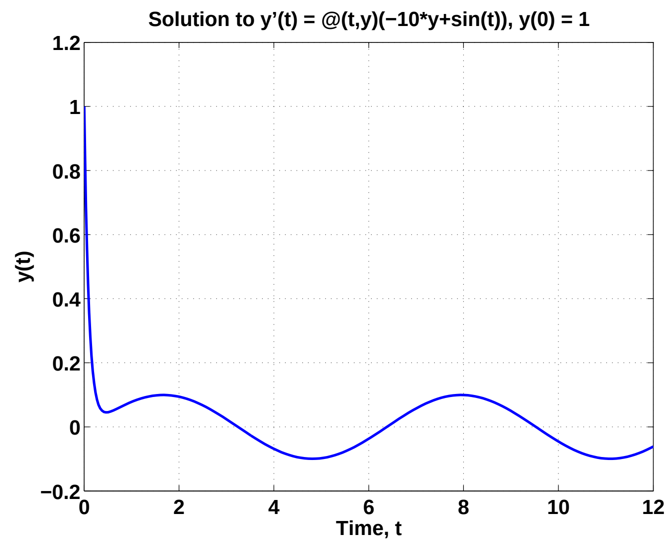
Results



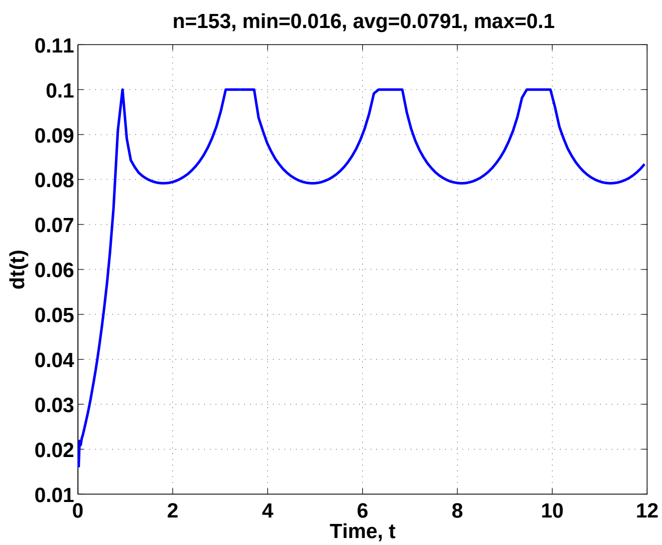
Results



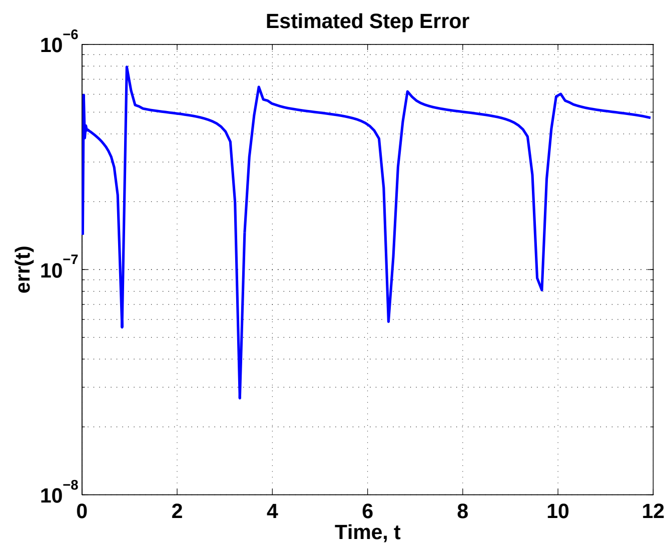
Results $y' = -10y + \sin(t)$



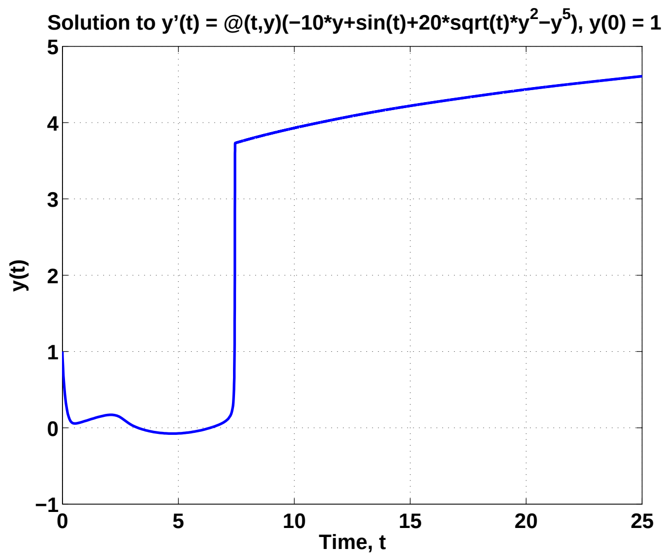
Results $y' = -10y + \sin(t)$



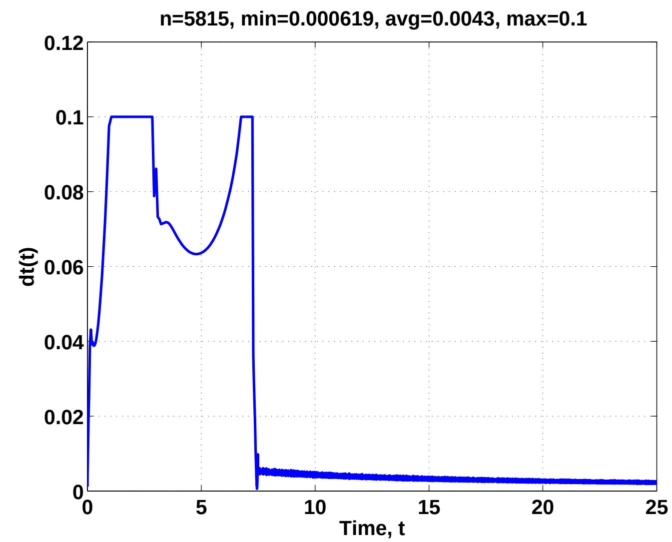
Results $y' = -10y + \sin(t)$



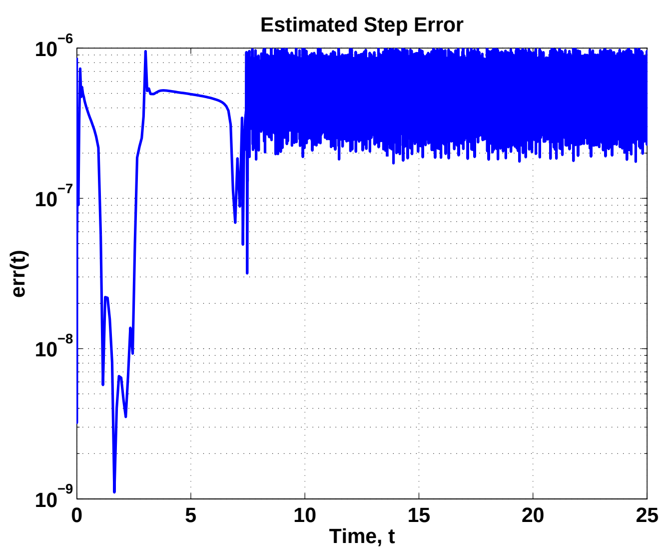
Results $y' = -10y + \sin(t) + 20\sqrt{t}\,y^2 - y^5$



Results $y' = -10y + \sin(t) + 20\sqrt{t}\,y^2 - y^5$



Results $y' = -10y + \sin(t) + 20\sqrt{t}\,y^2 - y^5$



RKF45, $s = 6$

$$
\begin{array}{c|c}
\tilde{\mathbf{c}} & A \\ \hline
 & \tilde{\mathbf{b}}^T \\
 & \tilde{\mathbf{b}}_2^T \\ \hline
 & \tilde{\mathbf{E}}^T
\end{array}
=
\begin{array}{c|cccccc}
0 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\
\frac{1}{4} & \frac{1}{4} & \ddots & & & & \vdots \\
\frac{3}{8} & \frac{3}{32} & \frac{9}{32} & \ddots & & & \vdots \\
\frac{12}{13} & \frac{1932}{2197} & -\frac{7200}{2197} & \frac{7296}{2197} & \ddots & & \vdots \\
1 & \frac{439}{216} & -8 & \frac{3680}{513} & -\frac{845}{4104} & \ddots & \vdots \\
\frac{1}{2} & -\frac{8}{27} & 2 & -\frac{3544}{2565} & \frac{1859}{4104} & -\frac{11}{40} & 0 \\ \hline
 & \frac{25}{216} & 0 & \frac{1408}{2565} & \frac{2197}{4104} & -\frac{1}{5} & 0 \\
 & \frac{16}{135} & 0 & \frac{6656}{12825} & \frac{28561}{56430} & -\frac{9}{50} & \frac{2}{55} \\ \hline
 & \frac{1}{360} & 0 & -\frac{128}{4275} & -\frac{2197}{75240} & \frac{1}{50} & \frac{2}{55}
\end{array}
$$

RKF56, $s = 8$

$\mathbf{0}$	0	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	0
$\frac{1}{6}$	$\frac{1}{6}$	$\ddots$						$\vdots$
$\frac{4}{15}$	$\frac{4}{75}$	$\frac{16}{75}$	$\ddots$					$\vdots$
$\frac{2}{3}$	$\frac{5}{6}$	$-\frac{8}{3}$	$\frac{5}{2}$	$\ddots$				$\vdots$
$\frac{4}{5}$	$-\frac{8}{5}$	$\frac{144}{25}$	-4	$\frac{16}{25}$	$\ddots$			$\vdots$
$\mathbf{1}$	$\frac{361}{320}$	$-\frac{18}{5}$	$\frac{407}{128}$	$-\frac{11}{80}$	$\frac{55}{128}$	$\ddots$		$\vdots$
$\mathbf{0}$	$-\frac{11}{640}$	0	$\frac{11}{256}$	$-\frac{11}{160}$	$\frac{11}{256}$	0	$\ddots$	$\vdots$
$\mathbf{1}$	$\frac{93}{640}$	$-\frac{18}{5}$	$\frac{803}{256}$	$-\frac{11}{160}$	$-\frac{99}{256}$	0	1	0
	$\frac{31}{384}$	0	$\frac{1125}{2816}$	$\frac{9}{32}$	$\frac{125}{768}$	$\frac{5}{66}$	0	0
	$\frac{7}{1408}$	0	$\frac{1125}{2816}$	$\frac{9}{32}$	$\frac{125}{768}$	$\frac{2}{55}$	$\frac{5}{66}$	$\frac{5}{66}$
	$-\frac{5}{66}$	0	0	0	0	$-\frac{5}{66}$	$\frac{5}{66}$	$\frac{5}{66}$

RKF78, $s = 13$

$\tilde{\mathbf{c}}^T$	0	$\frac{2}{27}$	$\frac{1}{9}$	$\frac{1}{6}$	$\frac{5}{12}$	$\frac{1}{2}$	$\frac{5}{6}$	$\frac{1}{6}$	$\frac{2}{3}$	$\frac{1}{3}$	1	0	1
$\tilde{\mathbf{b}}_1^T$	$\frac{41}{840}$	0	0	0	0	$\frac{34}{105}$	$\frac{9}{35}$	$\frac{9}{35}$	$\frac{9}{280}$	$\frac{9}{280}$	$\frac{41}{840}$	0	0
$\tilde{\mathbf{b}}_2^T$	0	0	0	0	0	$\frac{34}{105}$	$\frac{9}{35}$	$\frac{9}{35}$	$\frac{9}{280}$	$\frac{9}{280}$	0	$\frac{41}{840}$	$\frac{41}{840}$
	$-\frac{41}{840}$	0	0	0	0	0	0	0	0	0	$-\frac{41}{840}$	$\frac{41}{840}$	$\frac{41}{840}$

Problems with RKF56, (RKF67), and RKF78

The RKF56, 67, and 78 pairs derived by Fehlberg[1970] have been criticized for lack of computational robustness.

The two schemes rely on the same quadrature ("sample") points; *i.e.* that k_7^{RKF56}, k_8^{RKF56}, k_{12}^{RKF78}, and k_{13}^{RKF78} rely on the *same* values of c_i.

In cases where the ODE is approximately equal to a pure quadrature problem, then the error estimates will be too optimistic.

The methods of Verner[1978] (RKVmn) "fix" this problem.

The pair of methods $(A, \tilde{\mathbf{b}}_1, \tilde{\mathbf{c}}; p)$ and $(A, \tilde{\mathbf{b}}_2, \tilde{\mathbf{c}}; p+1)$ were intended to be used as order p methods with asymptotically correct error estimators (of order $p+1$).

In many practical implementations, the order $p+1$ method is propagated, even though the p order method does not provide as asymptotically correct error estimate.

When the higher order method is propagated, it makes sense to pay extra attention to the properties of this method. Dormand-and-Prince[1980] introduced methods, *e.g.* "RK5(4)7M" (5th order propagator, 4th order "error estimator"; 7-stage method) which are designed such that the $\|\circ\|_2$-norm of the vector of error coefficients is small.