

Runge-Kutta-Fehlberg Method

Rationale for Variable Step-Size Algorithms

Consider the following IVP, its solution, and the graph of the solution.

> **restart:**

> **deq:=D(y)(t)=-3*t*y(t)^2+1/(1+t^3);**

$$deq := D(y)(t) = -3ty(t)^2 + \frac{1}{1+t^3}$$

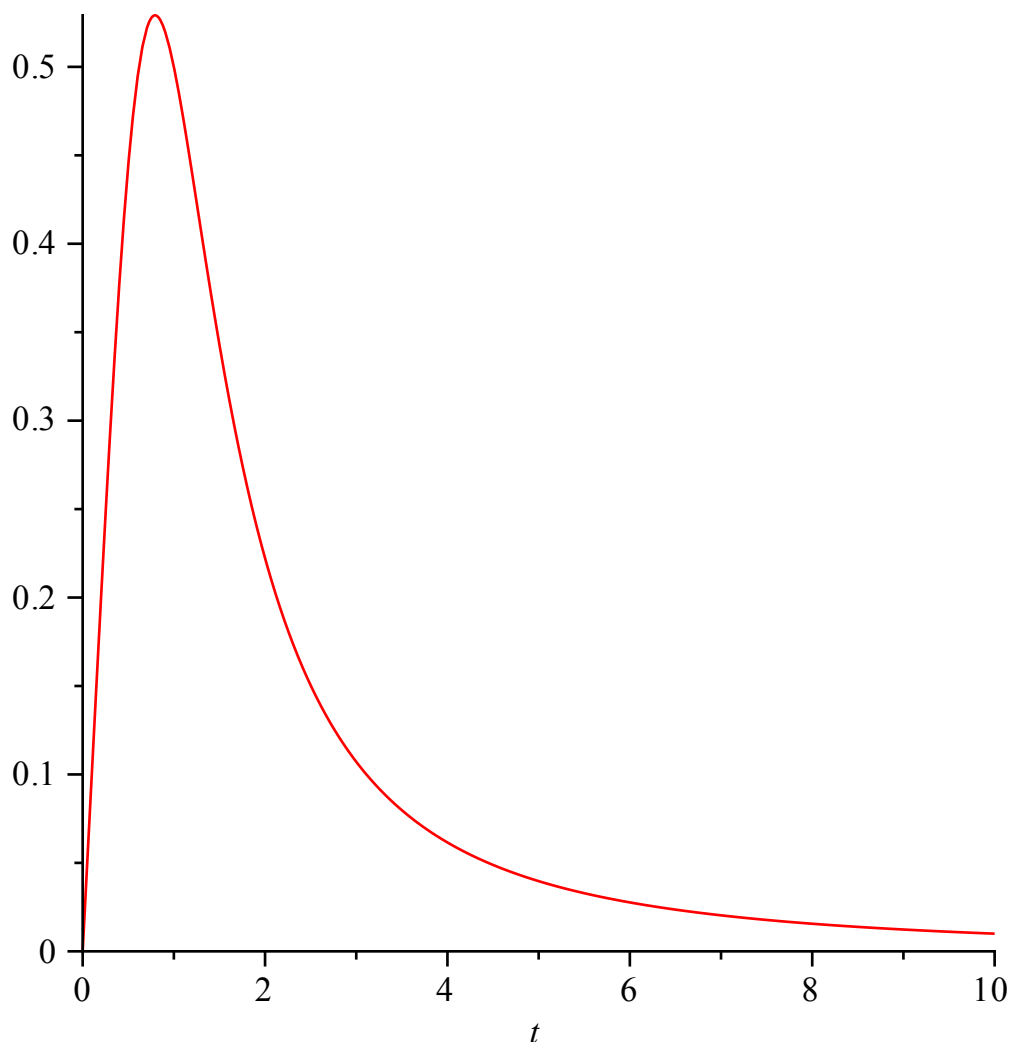
> **init:=y(0)=0;**

$$init := y(0) = 0$$

> **soln:=dsolve({deq,init},y(t));**

$$soln := y(t) = \frac{t}{1+t^3}$$

> **plot(rhs(soln),t=0..10);**



To accurately approximate the rapidly changing portion of this solution from 0 to 2.5 we need a fairly

small step-size. But using the same step-size in the flatter portions of the graph leads to way more work than necessary. Thus the rationale for variable step methods.

Runge-Kutta-Fehlberg method

The Runge-Kutta-Fehlberg method uses a Runge-Kutta method of order 5 to estimate the local error in a Runge-Kutta method of order 4. Working with a predetermined global error, the initial step size h is replaced by a step size qh at a given step if the error using h is not within the required bounds. We examine the method on the IVP

$$\frac{\partial}{\partial t} y = y - t^2 + 1, y(0) = 0.5 \text{ on } [0,5]$$

and compare it both with the exact solution and with Runge-Kutta's method of order 4 with $h = 0.1$.

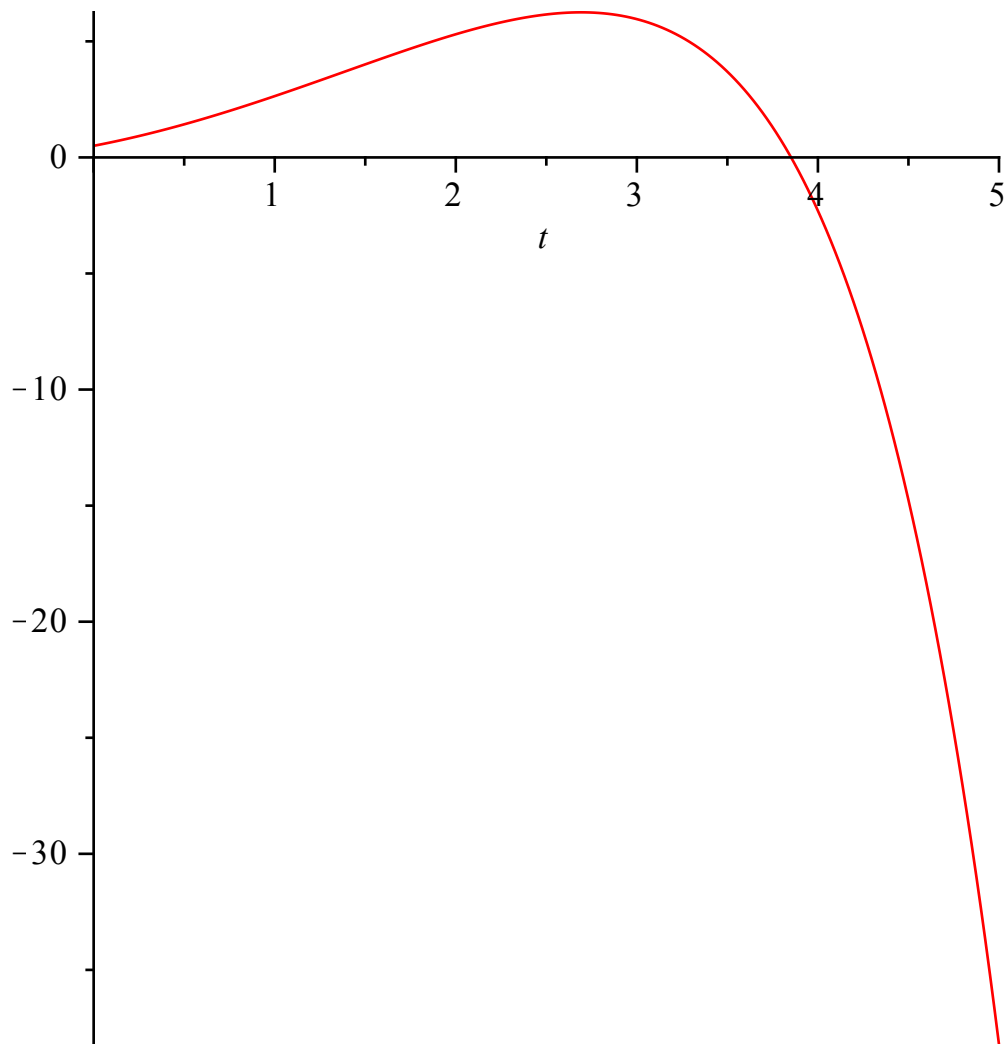
```
> restart:with(plots):
> deq:=D(y)(t)=y(t)-t*t+1;
      deq := D(y)(t) = y(t) - t^2 + 1
> init:=y(0)=0.5;
      init := y(0) = 0.5
```

We first find the exact solution, which we rewrite as a function Y .

```
> soln:=dsolve({deq,init},y(t));
      soln := y(t) = 1 + 2 t + t^2 - 1/2 e^t
> Y:=unapply(rhs(soln),t);
      Y := t -> 1 + 2 t + t^2 - 1/2 e^t
```

We plot the solution.

```
> plot(Y(t),t=0..5);
```



We see that the solution begins to change rapidly beginning with $t = 3$. We next apply **Runge-Kutta's** method of order 4 with a stepsize of $h = 0.1$.

```
> rk:=dsolve({deq, init}, y(t), type=numeric,
             method=classical[rk4], start=0.0, stepsize=.1);
             rk:=proc(x_classical) ... end proc
```

We use `dsolve` to implement the **Runge-Kutta-Fehberg** method by setting the type to **numeric**, the method to `rkf45`, and providing a starting point for the independent variable. Note that `rkf45` is the default numerical method in Maple, so `method=rkf45` need not be written. The absolute error is controlled by a statement of the type `abserr=Float(m,n)` where `Float(m,n)=m*10^n`, The default is `Float(1,-7)`, which equals 10^{-7} . The relative error is controlled by a statement of the type `relerr=Float(m,n)` where `Float(m,n)=m*10^n`. The default is `Float(1,-6)`, which equals 10^{-6} . The argument `initstep` gives an initial step size, and `range` gives the range of t values that we wish to use. `rkf45` uses a 4th degree interpolant for values between steps.

```
> rkf:=dsolve({deq, init}, y(t), type=numeric,
             method=rkf45, start=0.0, initstep=.1, range=0..5);
             rkf:=proc(x_rkf45) ... end proc
```

Now let's compare our results.

```
> printf("      t          y          rk(h=.1)      |y-rkf|
      rkf          |y-rkf|\n");
```

```

printf("\n");
for i from 0 to 50 do
printf("%4.1f %13.9f %13.9f %12.9f %13.9f %12.9f\n",.1*i,Y
(.1*i),eval(y(t),rk(.1*i)),abs(Y(.1*i)-eval(y(t),rk(.1*i))),eval
(y(t),rkf(.1*i)),abs(Y(.1*i)-eval(y(t),rkf(.1*i)))));
od;

```

t	y	rk(h=.1)	y-rk	rkf
0.0	0.500000000	0.500000000	0.000000000	0.500000000
0.1	0.657414541	0.657414375	0.000000166	0.657414540
0.2	0.829298621	0.829298276	0.000000345	0.829298691
0.3	1.015070596	1.015070058	0.000000538	1.015070636
0.4	1.214087651	1.214086906	0.000000745	1.214087654
0.5	1.425639364	1.425638396	0.000000968	1.425639459
0.6	1.648940600	1.648939390	0.000001210	1.648940534
0.7	1.883123646	1.883122179	0.000001467	1.883123770
0.8	2.127229536	2.127227791	0.000001745	2.127229417
0.9	2.380198444	2.380196402	0.000002042	2.380198586
1.0	2.640859086	2.640856724	0.000002362	2.640858930
1.1	2.907916988	2.907914285	0.000002703	2.907917095
1.2	3.179941538	3.179938470	0.000003068	3.179941525
1.3	3.455351666	3.455348207	0.000003459	3.455351547
1.4	3.732400016	3.732396141	0.000003875	3.732400481
1.5	4.009155465	4.009151145	0.000004320	4.009155199
1.6	4.283483788	4.283478996	0.000004792	4.283484092
1.7	4.553026304	4.553021009	0.000005295	4.553028709
1.8	4.815176268	4.815170440	0.000005828	4.815177429
1.9	5.067052779	5.067046386	0.000006393	5.067053312
2.0	5.305471950	5.305464960	0.000006990	5.305473264
2.1	5.526915044	5.526907423	0.000007621	5.526916133
2.2	5.727493250	5.727484966	0.000008284	5.727494595
2.3	5.902908772	5.902899791	0.000008981	5.902910860
2.4	6.048411810	6.048402098	0.000009712	6.048413629
2.5	6.158753020	6.158742545	0.000010475	6.158755862
2.6	6.228130980	6.228119714	0.000011266	6.228133817

```

0.000002837
 2.7   6.250134140      6.250122046   0.000012094   6.250137786
0.000003646
 2.8   6.217676615      6.217663673   0.000012942   6.217680601
0.000003986
 2.9   6.122927315      6.122913502   0.000013813   6.122932074
0.000004759
 3.0   5.957231540      5.957216834   0.000014706   5.957236774
0.000005234
 3.1   5.711024360      5.711008751   0.000015609   5.711030580
0.000006220
 3.2   5.373734900      5.373718384   0.000016516   5.373741570
0.000006670
 3.3   4.933680540      4.933663116   0.000017424   4.933688533
0.000007993
 3.4   4.377949980      4.377931660   0.000018320   4.377958538
0.000008558
 3.5   3.692274020      3.692254840   0.000019180   3.692283971
0.000009951
 3.6   2.860882780      2.860862775   0.000020005   2.860893887
0.000011107
 3.7   1.866347820      1.866327055   0.000020765   1.866360004
0.000012184
 3.8   0.689407760      0.689386310   0.000021450   0.689421457
0.000013697
 3.9   -0.691224560     -0.691246566   0.000022006   -0.691209270
0.000015290
 4.0   -2.299075020     -2.299097460   0.000022440   -2.299058113
0.000016907
 4.1   -4.160143800     -4.160166497   0.000022697   -4.160125069
0.000018731
 4.2   -6.303165520     -6.303188258   0.000022738   -6.303144756
0.000020764
 4.3   -8.759896850     -8.759919361   0.000022511   -8.759874037
0.000022813
 4.4  -11.565434330     -11.565456297   0.000021970  -11.565409331
0.000024999
 4.5  -14.758565650     -14.758586682   0.000021030  -14.758537174
0.000028476
 4.6  -18.382157820     -18.382177459   0.000019640  -18.382126257
0.000031563
 4.7  -22.483586250     -22.483603922   0.000017670  -22.483552085
0.000034165
 4.8  -27.115208750     -27.115223866   0.000015120  -27.115169197
0.000039553
 4.9  -32.334889850     -32.334901598   0.000011750  -32.334846281
0.000043569
 5.0  -38.206579550     -38.206587061   0.000007510  -38.206532171
0.000047379

```

Something funny is going on here. Beginning with $t = 4.3$, **rk4** is more accurate than **rkf45**. Could this be an issue with round-off error? We raise **Digits** to minimize the effect of round-off error.

```

> Digits:=25;
                                Digits := 25
> rk:=dsolve({deq, init}, y(t), type=numeric,
    method=classical[rk4], start=0.0,stepsize=.1);
    rk := proc(x_classical) ... end proc
> infolevel[dsolve]=5;
                                infoleveldsolve = 5

```

```

> rkf:=dsolve({deg, init}, y(t), type=numeric,
              method=rkf45, start=0.0, range=0..5);
              rkf:=proc(x_rkf45) ... end proc
> printf("    t          y          rk(h=.1)    |y-rkf|
rkf          |y-rkf|\n");
printf("\n");
for i from 0 to 50 do
printf("%4.1f %13.9f %13.9f %12.9f %13.9f %12.9f\n",.1*i,Y
(.1*i),eval(y(t),rk(.1*i)),abs(Y(.1*i)-eval(y(t),rk(.1*i))),eval
(y(t),rkf(.1*i)),abs(Y(.1*i)-eval(y(t),rkf(.1*i)))));
od;

```

t	y	rk(h=.1)	y-rkf	rkf
0.0	0.500000000	0.500000000	0.000000000	0.500000000
0.1	0.657414541	0.657414375	0.000000166	0.657414616
0.2	0.829298621	0.829298276	0.000000345	0.829298630
0.3	1.015070596	1.015070058	0.000000538	1.015070622
0.4	1.214087651	1.214086906	0.000000745	1.214087721
0.5	1.425639365	1.425638396	0.000000969	1.425639315
0.6	1.648940600	1.648939390	0.000001209	1.648940706
0.7	1.883123646	1.883122179	0.000001468	1.883123543
0.8	2.127229536	2.127227791	0.000001745	2.127229663
0.9	2.380198444	2.380196402	0.000002043	2.380198283
1.0	2.640859086	2.640856724	0.000002362	2.640859223
1.1	2.907916988	2.907914285	0.000002703	2.907916822
1.2	3.179941539	3.179938470	0.000003068	3.179941580
1.3	3.455351666	3.455348207	0.000003459	3.455351805
1.4	3.732400017	3.732396141	0.000003875	3.732399744
1.5	4.009155465	4.009151145	0.000004320	4.009156096
1.6	4.283483788	4.283478996	0.000004792	4.283484099
1.7	4.553026304	4.553021009	0.000005295	4.553026068
1.8	4.815176268	4.815170440	0.000005828	4.815176988
1.9	5.067052779	5.067046386	0.000006393	5.067053158
2.0	5.305471951	5.305464960	0.000006990	5.305471986
2.1	5.526915044	5.526907423	0.000007621	5.526916429
2.2	5.727493250	5.727484966	0.000008285	5.727494385
2.3	5.902908773	5.902899791	0.000008982	5.902909888

```

0.000001115
 2.4   6.048411810      6.048402098   0.000009712   6.048413852
0.000002042
 2.5   6.158753020      6.158742545   0.000010475   6.158754791
0.000001771
 2.6   6.228130982      6.228119714   0.000011269   6.228133708
0.000002725
 2.7   6.250134138      6.250122046   0.000012092   6.250136877
0.000002739
 2.8   6.217676614      6.217663673   0.000012941   6.217680181
0.000003566
 2.9   6.122927315      6.122913502   0.000013814   6.122931119
0.000003804
 3.0   5.957231538      5.957216834   0.000014705   5.957236240
0.000004701
 3.1   5.711024359      5.711008751   0.000015609   5.711029345
0.000004986
 3.2   5.373734901      5.373718384   0.000016518   5.373741032
0.000006131
 3.3   4.933680540      4.933663116   0.000017424   4.933687003
0.000006463
 3.4   4.377949976      4.377931660   0.000018316   4.377957743
0.000007767
 3.5   3.692274021      3.692254840   0.000019181   3.692282530
0.000008510
 3.6   2.860882778      2.860862775   0.000020004   2.860892229
0.000009451
 3.7   1.866347820      1.866327055   0.000020765   1.866358657
0.000010837
 3.8   0.689407753      0.689386310   0.000021443   0.689419751
0.000011997
 3.9   -0.691224553     -0.691246566   0.000022013   -0.691211260
0.000013293
 4.0   -2.299075017     -2.299097460   0.000022443   -2.299060288
0.000014729
 4.1   -4.160143799     -4.160166497   0.000022698   -4.160127563
0.000016236
 4.2   -6.303165520     -6.303188258   0.000022737   -6.303147695
0.000017825
 4.3   -8.759896850     -8.759919361   0.000022511   -8.759876982
0.000019868
 4.4  -11.565434332     -11.565456297   0.000021964  -11.565411902
0.000022430
 4.5  -14.758565650     -14.758586682   0.000021032  -14.758541122
0.000024528
 4.6  -18.382157821     -18.382177459   0.000019638  -18.382130947
0.000026874
 4.7  -22.483586226     -22.483603922   0.000017696  -22.483555057
0.000031169
 4.8  -27.115208759     -27.115223866   0.000015107  -27.115174871
0.000033888
 4.9  -32.334889842     -32.334901598   0.000011755  -32.334852406
0.000037436
 5.0  -38.206579551     -38.206587061   0.000007510  -38.206535832
0.000043719

```

Still more error with **rkf45**. Now we try **relerr** = 10^{-9} . From the [help](#) pages, $approxerr_i \leq abserr + relerr |y_i|$, where $approxerr_i$ is what we think of as global error.

```

> rkf:=dsolve({deq, init}, y(t), type=numeric,
method=rkf45, start=0.0, range=0..5, relerr=Float(1,-9));
rkf:=proc(x_rkf45) ... end proc

```

```

> printf("    t          y          rk(h=.1)      |y-rk|
rkf          |y-rkf|\n");
printf("\n");
for i from 0 to 50 do
printf("%4.1f %13.9f %13.9f %12.9f %13.9f %12.9f\n",.1*i,Y
(.1*i),eval(y(t),rk(.1*i)),abs(Y(.1*i)-eval(y(t),rk(.1*i))),eval
(y(t),rkf(.1*i)),abs(Y(.1*i)-eval(y(t),rkf(.1*i)))));
od;

```

t	y	rk(h=.1)	y-rk	rkf
0.0	0.500000000	0.500000000	0.000000000	0.500000000
0.1	0.657414541	0.657414375	0.000000166	0.657414540
0.2	0.829298621	0.829298276	0.000000345	0.829298621
0.3	1.015070596	1.015070058	0.000000538	1.015070598
0.4	1.214087651	1.214086906	0.000000745	1.214087656
0.5	1.425639365	1.425638396	0.000000969	1.425639370
0.6	1.648940600	1.648939390	0.000001209	1.648940604
0.7	1.883123646	1.883122179	0.000001468	1.883123646
0.8	2.127229536	2.127227791	0.000001745	2.127229528
0.9	2.380198444	2.380196402	0.000002043	2.380198428
1.0	2.640859086	2.640856724	0.000002362	2.640859068
1.1	2.907916988	2.907914285	0.000002703	2.907916981
1.2	3.179941539	3.179938470	0.000003068	3.179941537
1.3	3.455351666	3.455348207	0.000003459	3.455351648
1.4	3.732400017	3.732396141	0.000003875	3.732399989
1.5	4.009155465	4.009151145	0.000004320	4.009155508
1.6	4.283483788	4.283478996	0.000004792	4.283483762
1.7	4.553026304	4.553021009	0.000005295	4.553026284
1.8	4.815176268	4.815170440	0.000005828	4.815176265
1.9	5.067052779	5.067046386	0.000006393	5.067052761
2.0	5.305471951	5.305464960	0.000006990	5.305471928
2.1	5.526915044	5.526907423	0.000007621	5.526915024
2.2	5.727493250	5.727484966	0.000008285	5.727493233
2.3	5.902908773	5.902899791	0.000008982	5.902908757
2.4	6.048411810	6.048402098	0.000009712	6.048411795
2.5	6.158753020	6.158742545	0.000010475	6.158753004

0.000000015				
2.6	6.228130982	6.228119714	0.000011269	6.228130968
0.000000015				
2.7	6.250134138	6.250122046	0.000012092	6.250134126
0.000000012				
2.8	6.217676614	6.217663673	0.000012941	6.217676608
0.000000007				
2.9	6.122927315	6.122913502	0.000013814	6.122927313
0.000000002				
3.0	5.957231538	5.957216834	0.000014705	5.957231536
0.000000003				
3.1	5.711024359	5.711008751	0.000015609	5.711024354
0.000000005				
3.2	5.373734901	5.373718384	0.000016518	5.373734904
0.000000002				
3.3	4.933680540	4.933663116	0.000017424	4.933680547
0.000000007				
3.4	4.377949976	4.377931660	0.000018316	4.377949979
0.000000003				
3.5	3.692274021	3.692254840	0.000019181	3.692274034
0.000000013				
3.6	2.860882778	2.860862775	0.000020004	2.860882789
0.000000011				
3.7	1.866347820	1.866327055	0.000020765	1.866347840
0.000000020				
3.8	0.689407753	0.689386310	0.000021443	0.689407773
0.000000019				
3.9	-0.691224553	-0.691246566	0.000022013	-0.691224522
0.000000030				
4.0	-2.299075017	-2.299097460	0.000022443	-2.299074988
0.000000029				
4.1	-4.160143799	-4.160166497	0.000022698	-4.160143758
0.000000041				
4.2	-6.303165520	-6.303188258	0.000022737	-6.303165474
0.000000046				
4.3	-8.759896850	-8.759919361	0.000022511	-8.759896802
0.000000048				
4.4	-11.565434332	-11.565456297	0.000021964	-11.565434271
0.000000061				
4.5	-14.758565650	-14.758586682	0.000021032	-14.758565577
0.000000073				
4.6	-18.382157821	-18.382177459	0.000019638	-18.382157742
0.000000079				
4.7	-22.483586226	-22.483603922	0.000017696	-22.483586140
0.000000086				
4.8	-27.115208759	-27.115223866	0.000015107	-27.115208660
0.000000099				
4.9	-32.334889842	-32.334901598	0.000011755	-32.334889724
0.000000118				
5.0	-38.206579551	-38.206587061	0.000007510	-38.206579415
0.000000137				

Now **rkf45** is clearly superior.

```
> with(Student[NumericalAnalysis]);
```

```
[AbsoluteError, AdamsBashforth, AdamsBashforthMoulton, AdamsMoulton, AdaptiveQuadrature,
AddPoint, ApproximateExactUpperBound, ApproximateValue, BackSubstitution, BasisFunctions,
Bisection, CubicSpline, DataPoints, Distance, DividedDifferenceTable, Draw, Euler, EulerTutor,
ExactValue, FalsePosition, FixedPointIteration, ForwardSubstitution, Function,
```

InitialValueProblem, InitialValueProblemTutor, Interpolant, InterpolantRemainderTerm, IsConvergent, IsMatrixShape, IterativeApproximate, IterativeFormula, IterativeFormulaTutor, LeadingPrincipalSubmatrix, LinearSolve, LinearSystem, MatrixConvergence, MatrixDecomposition, MatrixDecompositionTutor, ModifiedNewton, NevilleTable, Newton, NumberOfSignificantDigits, PolynomialInterpolation, Quadrature, RateOfConvergence, RelativeError, RemainderTerm, Roots, RungeKutta, Secant, SpectralRadius, Steffensen, Taylor, TaylorPolynomial, UpperBoundOfRemainderTerm, VectorLimit]

We will solve the same problem with which we have been orking by using the [RungeKutta](#) command, which is short for [InitialValueProblem](#) with **method=rungekutta**. We will also use **submethod= rkf**. We re-enter the IVP after resetting some variables that have been given values.

```
> y=`y`;t=`t`;
```

y = y

t = t

```
> deq:=diff(y(t),t)=y(t)-t^2+1;
```

$$deq := \frac{d}{dt} y(t) = y(t) - t^2 + 1$$

```
> init:=y(0)=0.5;
```

init := y(0) = 0.5

We reset the **interface** so that we get full matrix display.

```
> interface(rtablesize=100);
```

10

```
> InitialValueProblem(deq,y(0)=0.5,t=5,method=rungekutta,submethod= rkf,numsteps=50,digits=10,output=information);
```

t	$y(t)$	[<i>Runge-Kutta Fehlberg</i>]	[<i>Error</i>]
0.	0.5	0.5000000000	0.
0.1000000000	0.6574145410	0.6574146157	$7.47 \cdot 10^{-8}$
0.2000000000	0.8292986209	0.8292986296	$8.7 \cdot 10^{-9}$
0.3000000000	1.015070596	1.015070622	$2.6 \cdot 10^{-8}$
0.4000000000	1.214087651	1.214087721	$7.0 \cdot 10^{-8}$
0.5000000000	1.425639365	1.425639315	$5.0 \cdot 10^{-8}$
0.6000000000	1.648940600	1.648940706	$1.06 \cdot 10^{-7}$
0.7000000000	1.883123646	1.883123543	$1.03 \cdot 10^{-7}$
0.8000000000	2.127229536	2.127229663	$1.27 \cdot 10^{-7}$
0.9000000000	2.380198444	2.380198283	$1.61 \cdot 10^{-7}$
1.	2.640859086	2.640859223	$1.37 \cdot 10^{-7}$
1.1000000000	2.907916988	2.907916822	$1.66 \cdot 10^{-7}$
1.2000000000	3.179941539	3.179941580	$4.1 \cdot 10^{-8}$
1.3000000000	3.455351666	3.455351805	$1.39 \cdot 10^{-7}$
1.4000000000	3.732400017	3.732399744	$2.73 \cdot 10^{-7}$
1.5000000000	4.009155465	4.009156096	$6.31 \cdot 10^{-7}$
1.6000000000	4.283483788	4.283484099	$3.11 \cdot 10^{-7}$
1.7000000000	4.553026304	4.553026068	$2.36 \cdot 10^{-7}$
1.8000000000	4.815176268	4.815176988	$7.20 \cdot 10^{-7}$
1.9000000000	5.067052779	5.067053158	$3.79 \cdot 10^{-7}$
2.	5.305471951	5.305471986	$3.5 \cdot 10^{-8}$
2.1000000000	5.526915044	5.526916429	0.000001385
2.2000000000	5.727493250	5.727494385	0.000001135
2.3000000000	5.902908773	5.902909888	0.000001115
2.4000000000	6.048411810	6.048413852	0.000002042
2.5000000000	6.158753020	6.158754791	0.000001771
2.6000000000	6.228130982	6.228133708	0.000002726
2.7000000000	6.250134138	6.250136877	0.000002739
2.8000000000	6.217676614	6.217680181	0.000003567
2.9000000000	6.122927315	6.122931119	0.000003804

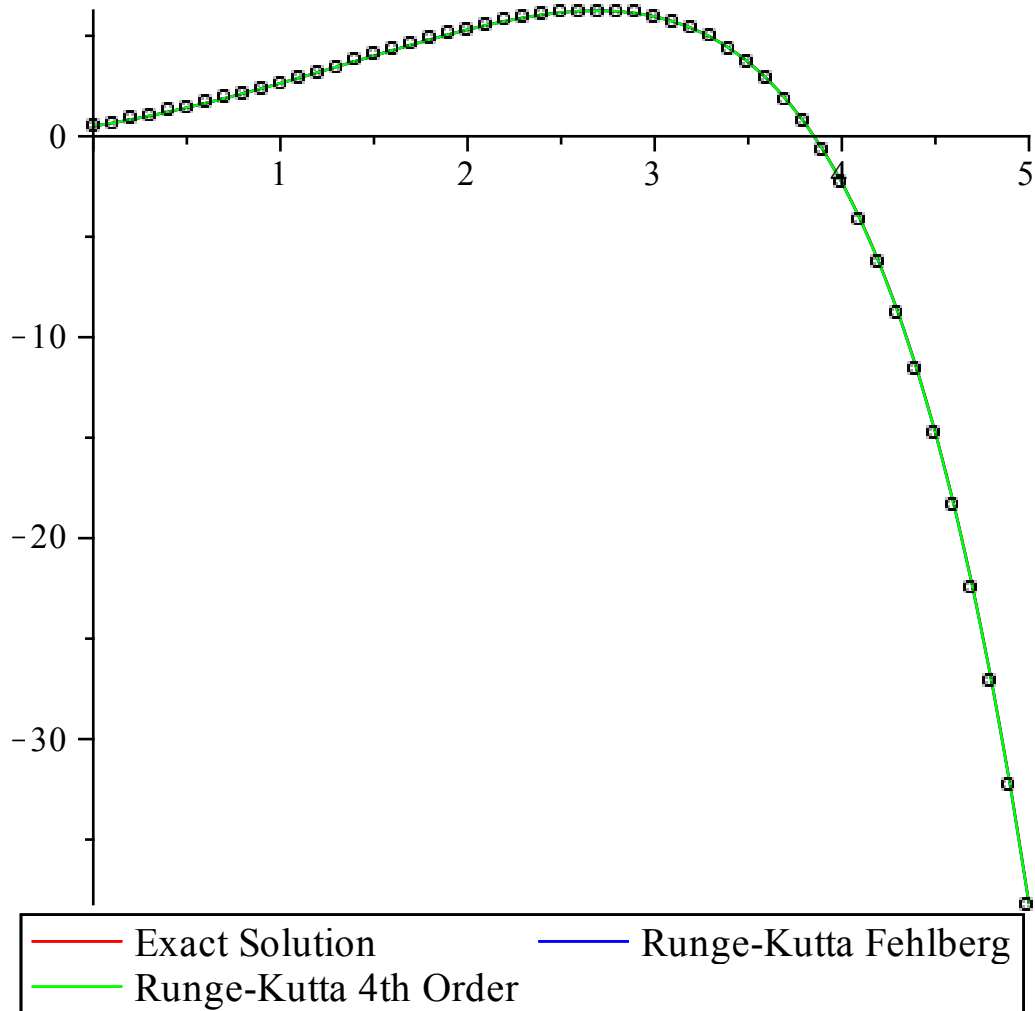
We have very good accuracy here. We use the argument `comparewith=[[rungekutta,rk4]]` to compare **Runge-Kuta-Fehlberg** with the **Fourth Order Runge-Kutta** method. This is available when `output=information or plot`. We start with information.

```
> RungeKutta(deq, y(0)=0.5, t=5, submethod=rkf, numsteps=50, digits=10,  
output=information, comparewith=[ [rungekutta, rk4] ] );
```

t	$y(t)$	[$R-K-F$]	[$Error$]	[$R-K$ 4th Ord.]	[$Error$]
0.	0.5	0.5000000000	0.	0.5	0.
0.1000000000	0.6574145410	0.6574146157	$7.47 \cdot 10^{-8}$	0.6574143750	$1.660 \cdot 10^{-7}$
0.2000000000	0.8292986209	0.8292986296	$8.7 \cdot 10^{-9}$	0.8292982760	$3.449 \cdot 10^{-7}$
0.3000000000	1.015070596	1.015070622	$2.6 \cdot 10^{-8}$	1.015070058	$5.38 \cdot 10^{-7}$
0.4000000000	1.214087651	1.214087721	$7.0 \cdot 10^{-8}$	1.214086906	$7.45 \cdot 10^{-7}$
0.5000000000	1.425639365	1.425639315	$5.0 \cdot 10^{-8}$	1.425638396	$9.69 \cdot 10^{-7}$
0.6000000000	1.648940600	1.648940706	$1.06 \cdot 10^{-7}$	1.648939390	0.000001210
0.7000000000	1.883123646	1.883123543	$1.03 \cdot 10^{-7}$	1.883122179	0.000001467
0.8000000000	2.127229536	2.127229663	$1.27 \cdot 10^{-7}$	2.127227791	0.000001745
0.9000000000	2.380198444	2.380198283	$1.61 \cdot 10^{-7}$	2.380196402	0.000002042
1.	2.640859086	2.640859223	$1.37 \cdot 10^{-7}$	2.640856724	0.000002362
1.1000000000	2.907916988	2.907916822	$1.66 \cdot 10^{-7}$	2.907914285	0.000002703
1.2000000000	3.179941539	3.179941580	$4.1 \cdot 10^{-8}$	3.179938470	0.000003069
1.3000000000	3.455351666	3.455351805	$1.39 \cdot 10^{-7}$	3.455348207	0.000003459
1.4000000000	3.732400017	3.732399744	$2.73 \cdot 10^{-7}$	3.732396141	0.000003876
1.5000000000	4.009155465	4.009156096	$6.31 \cdot 10^{-7}$	4.009151145	0.000004320
1.6000000000	4.283483788	4.283484099	$3.11 \cdot 10^{-7}$	4.283478996	0.000004792
1.7000000000	4.553026304	4.553026068	$2.36 \cdot 10^{-7}$	4.553021009	0.000005295
1.8000000000	4.815176268	4.815176988	$7.20 \cdot 10^{-7}$	4.815170440	0.000005828
1.9000000000	5.067052779	5.067053158	$3.79 \cdot 10^{-7}$	5.067046386	0.000006393
2.	5.305471951	5.305471986	$3.5 \cdot 10^{-8}$	5.305464960	0.000006991
2.1000000000	5.526915044	5.526916429	0.000001385	5.526907423	0.000007621
2.2000000000	5.727493250	5.727494385	0.000001135	5.727484966	0.000008284
2.3000000000	5.902908773	5.902909888	0.000001115	5.902899791	0.000008982
2.4000000000	6.048411810	6.048413852	0.000002042	6.048402098	0.000009712
2.5000000000	6.158753020	6.158754791	0.000001771	6.158742545	0.000010475
2.6000000000	6.228130982	6.228133708	0.000002726	6.228119714	0.000011268
2.7000000000	6.250134138	6.250136877	0.000002739	6.250122046	0.000012092
2.8000000000	6.217676614	6.217680181	0.000003567	6.217663673	0.000012941
2.9000000000	6.122927315	6.122931119	0.000003804	6.122913502	0.000013813

Starting at about 4.4, the **Fourth Order Runge-Kutta** method is the more accurate. But here, there is no option to request **absolute** and **relative** error levels as there is in the built-in **rkf45**. Now we look at **plot**.

```
> RungeKutta(deq, y(0)=0.5, t=5, submethod=rkf, numsteps=50, digits=10,  
output=plot, comparewith=[ [rungekutta, rk4] ] );
```



[It is clearly hard to distinguish anything here.