

Runge-Kutta Method of Order 4

This method is easier to compute than the Taylor method of order 4, and has the same order of local truncation error, $O(h^4)$. We examine the method on the IVP

$$\frac{\partial}{\partial t} y = y - t^2 + 1, y(0) = 0.5 \text{ on } [0,2], h = 0.1,$$

and compare it with the exact solution, Euler's method with $h = 0.025$, and the Modified Euler's Method with $h = 0.05$.

```
> 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 next apply Euler's method.

```
> eul:=dsolve({deq,init},y(t),type=numeric,
      method=classical[foreuler],start=0.0,stepsize=.025);
      eul := proc(x_classical) ... end proc
```

Now we apply the Modified Euler Method by setting the type to numeric, the method to **classical[heunform]**, and providing a starting point for the independent variable and a stepsize..

```
> modified:=dsolve({deq,init},y(t),type=numeric,
      method=classical[heunform],start=0.0,stepsize=.05);
      modified := proc(x_classical) ... end proc
```

We use `dsolve` to implement the Runge-Kutta method of order 4 by setting the type to numeric, the method to **classical[rk4]**, and providing a starting point for the independent variable and a stepsize.

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

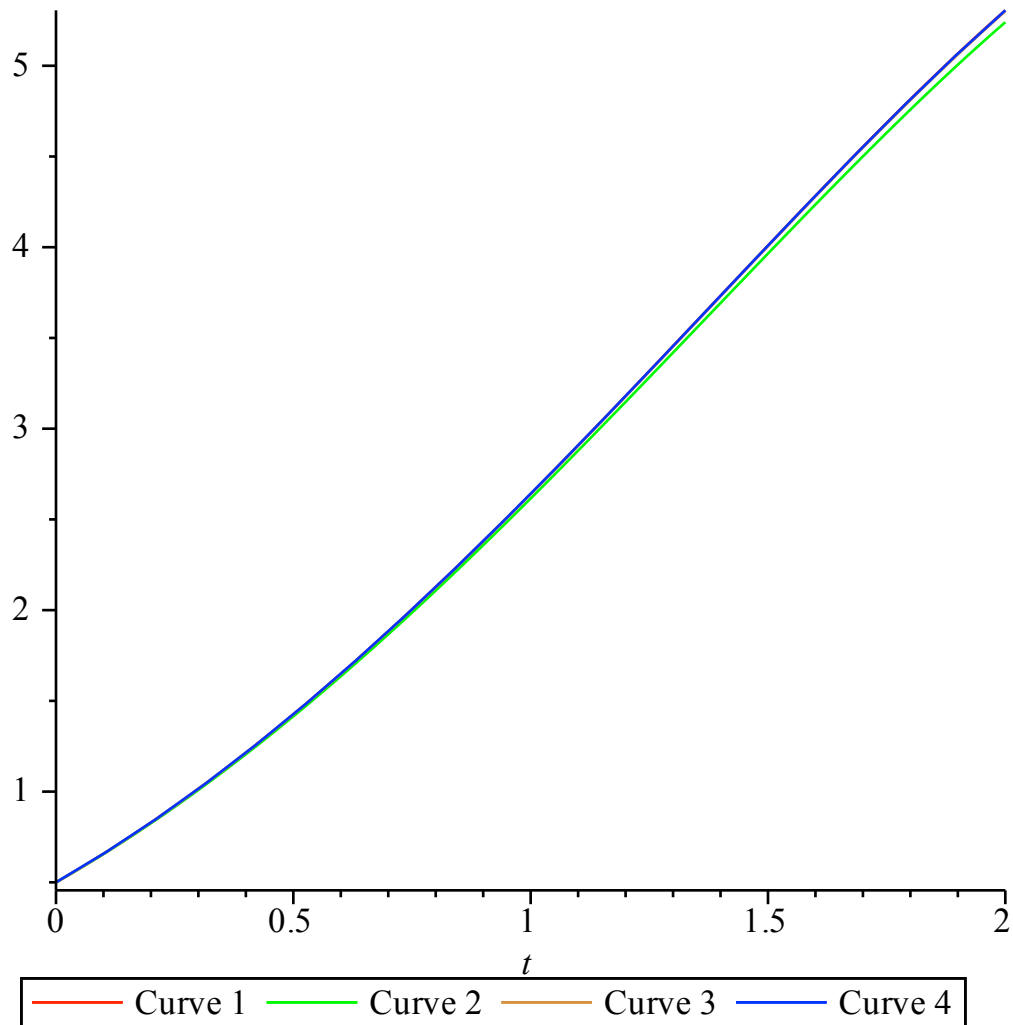
Now let's compare our results.

```
> printf("  t          y          eul(h=.025)      |y-eul|      mod(h=
      .05)      |y-modified|      rk(h=.1)      |y-rk|\n");
printf("\n");
for i from 0 to 20 do
printf("%4.1f  %12.7f  %12.7f  %12.7f  %12.7f  %12.7f  %12.7f
      %12.7f\n",.1*i,Y(.1*i),eval(y(t),eul(.1*i)),abs(Y(.1*i)-eval(y(t),
      eul(.1*i))),eval(y(t),modified(.1*i)),abs(Y(.1*i)-eval(y(t),
      modified(.1*i))),eval(y(t),rk(.1*i)),abs(Y(.1*i)-eval(y(t),rk(.1*i)
      ))));
od;
      t          y          eul(h=.025)      |y-eul|      mod(h=.05)      |y-
```

modified	rk(h=.1)	y-rk		
0.0	0.5000000	0.5000000	0.0000000	0.5000000
0.0000000	0.5000000	0.0000000		
0.1	0.6574145	0.6554982	0.0019163	0.6573085
0.0001060	0.6574144	0.0000002		
0.2	0.8292986	0.8253385	0.0039601	0.8290778
0.0002209	0.8292983	0.0000003		
0.3	1.0150706	1.0089334	0.0061372	1.0147254
0.0003452	1.0150701	0.0000005		
0.4	1.2140877	1.2056345	0.0084531	1.2136079
0.0004798	1.2140869	0.0000007		
0.5	1.4256394	1.4147264	0.0109130	1.4250141
0.0006253	1.4256384	0.0000010		
0.6	1.6489406	1.6354189	0.0135217	1.6481579
0.0007827	1.6489394	0.0000012		
0.7	1.8831236	1.8668401	0.0162835	1.8821709
0.0009528	1.8831222	0.0000015		
0.8	2.1272295	2.1080276	0.0192019	2.1260931
0.0011365	2.1272278	0.0000017		
0.9	2.3801984	2.3579190	0.0222795	2.3788637
0.0013348	2.3801964	0.0000020		
1.0	2.6408591	2.6153415	0.0255176	2.6393103
0.0015487	2.6408567	0.0000024		
1.1	2.9079170	2.8790008	0.0289162	2.9061375
0.0017795	2.9079143	0.0000027		
1.2	3.1799415	3.1474680	0.0324736	3.1779134
0.0020281	3.1799385	0.0000031		
1.3	3.4553517	3.4191660	0.0361856	3.4530558
0.0022959	3.4553482	0.0000035		
1.4	3.7324000	3.6923540	0.0400460	3.7298159
0.0025841	3.7323961	0.0000039		
1.5	4.0091555	3.9651104	0.0440451	4.0062614
0.0028940	4.0091511	0.0000043		
1.6	4.2834838	4.2353141	0.0481697	4.2802567
0.0032271	4.2834790	0.0000048		
1.7	4.5530263	4.5006238	0.0524025	4.5494416
0.0035847	4.5530210	0.0000053		
1.8	4.8151763	4.7584553	0.0567210	4.8112079
0.0039683	4.8151704	0.0000058		
1.9	5.0670528	5.0059559	0.0610969	5.0626732
0.0043795	5.0670464	0.0000064		
2.0	5.3054720	5.2399769	0.0654951	5.3006521
0.0048199	5.3054650	0.0000070		

Since for each value of t , all three methods require the same number of evaluations, it is clear that the Runge-Kutta method of order 4 is superior. Finally, let's look at the graphs of the four solutions.

```
> p[0]:=plot(Y(t),t=0..2):
p[1]:=odeplot(eul,[t,y(t)],0..2, numpoints=80,color=green):
p[2]:=odeplot(modified,[t,y(t)],0..2, numpoints=40,color=gold):
p[3]:=odeplot(rk,[t,y(t)],0..2, numpoints=20,color=blue):
display(p[0],p[1],p[2],p[3]);
```



We check to see whether intermediate values are computed by linear interpolation. We first check the value given by the procedure.

```
> eval(y(t), rk(1.97));
5.23555491173119414
```

Now we check the value given by linear interpolation.

```
> eval(y(t), rk(1.9)) + .7 * (eval(y(t), rk(2.0)) - eval(y(t), rk(1.9)));
5.233939388
```

Clearly, linear interpolation is not used. A good choice for intermediate values would be cubic Hermite interpolation. We load the library and review the directions.

```
> libname := "c:/nalib", libname;
libname := "/nalib", "/Library/Frameworks/Maple.framework/Versions/15/lib",
"/Library/Frameworks/Maple.framework/Versions/15/toolbox/NAG/lib"
> with(numanal);
[SOR, SOR_dir, adaptq, adaptq_dir, bezier, bezier_dir, bisection, bisection_dir, chop, chop_dir,
clamped_spline, clamped_spline_dir, divided_diff, divided_diff_dir, extrap, extrap_dir,
falseposition, falseposition_dir, fixedpoint, fixedpoint_dir, gaussseidel, gaussseidel_dir, hermite,
hermite_dd, hermite_dd_dir, hermite_dir, horner, horner_dir, jacobi, jacobi_dir, muller,
```

muller_dir, natural_spline, natural_spline_dir, newton, newton_dir, romberg, romberg_dir, secant, secant_dir, steffensen, steffensen_dir]

```
> hermite_dir();  
hermite returns the interpolating polynomial.
```

The arguments for hermite are:

- (1) the list of x-values
- (2) the list of f(x) or y-values
- (3) the list of f'(x)-values
- (4) the variable for returning the polynomial

If assigning the result to a variable, have the variable and the 4th argument the same.

If p is the variable for returning the polynomial and has already been given a value, the procedure should be preceded by the statement:
p:='p'

We turn the right hand side of the differential equation into a function of two variables so as to be able to compute the list of derivatives.

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

$$f := (t, y) \rightarrow y - t^2 + 1$$

In order, we compute the lists of x values, y values, and derivatives, using the approximations from the table for the last two.

```
> XX:=[1.9,2.0];
```

$$XX := [1.9, 2.0]$$

```
> YY:=[evalf(eval(y(t),rk(1.9))),evalf(eval(y(t),rk(2.0)))];  
YY := [5.067046386, 5.305464960]
```

```
> ZZ:=[f(1.9,eval(y(t),rk(1.9))),f(2.0,eval(y(t),rk(2.0)))];  
ZZ := [2.457046386, 2.305464960]
```

We compute the Hermite cubic polynomial.

```
> p:=hermite(XX,YY,ZZ,p);
```

The interpolating polynomial is $1.999409 - 0.58601x^3 + 2.67026x^2 - 1.3435x$

$$p := 1.999409 - 0.58601 x^3 + 2.67026 x^2 - 1.3435 x$$

We evaluate this polynomial at $t = 1.97$.

```
> subs(x=1.97,p);
```

$$5.235460998$$

This value appears very close to the one given by Maple, so possibly Maple uses cubic Hermite interpolation also.

NumericalAnalysis

We now look to the **Student[NumericalAnalysis]** package.

```
> 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 began the worksheet by using the [RungeKutta](#) command, which is short for [InitialValueProblem](#) with **method=rungekutta**. We 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*t+1;
```

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

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

$init := y(0) = 0.5$

This method has several submethods: **midpoint** for Midpoint method, **meuler** for Modified Euler method, **heun** for Heun's method, **rk3** for Order Three method (not covered in class), **rk4** for Order Four method, and **rkf** for Runge-Kutta-Fehlberg method (to be covered later in class). We look at **rk4** with a step-size of 0.1 after first resetting interface.

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

10

```
> RungeKutta(deq,y(0)=0.5,t=2,submethod=rk4,numsteps=20,digits=10, output=information);
```

t	$y(t)$	[Runge-Kutta 4th Order]	[Error]
0.	0.5	0.5	0.
0.1000000000	0.6574145410	0.6574143750	$1.660 \cdot 10^{-7}$
0.2000000000	0.8292986209	0.8292982760	$3.449 \cdot 10^{-7}$
0.3000000000	1.015070596	1.015070058	$5.38 \cdot 10^{-7}$
0.4000000000	1.214087651	1.214086906	$7.45 \cdot 10^{-7}$
0.5000000000	1.425639365	1.425638396	$9.69 \cdot 10^{-7}$
0.6000000000	1.648940600	1.648939390	0.000001210
0.7000000000	1.883123646	1.883122179	0.000001467
0.8000000000	2.127229536	2.127227791	0.000001745
0.9000000000	2.380198444	2.380196402	0.000002042
1.	2.640859086	2.640856724	0.000002362
1.100000000	2.907916988	2.907914285	0.000002703
1.200000000	3.179941539	3.179938470	0.000003069
1.300000000	3.455351666	3.455348207	0.000003459
1.400000000	3.732400017	3.732396141	0.000003876
1.500000000	4.009155465	4.009151145	0.000004320
1.600000000	4.283483788	4.283478996	0.000004792
1.700000000	4.553026304	4.553021009	0.000005295
1.800000000	4.815176268	4.815170440	0.000005828
1.900000000	5.067052779	5.067046386	0.000006393
2.	5.305471951	5.305464960	0.000006991

We have very good accuracy here. We use the argument `comparewith=[[rungekutta,midpoint],[rungekutta,meuler],[rungekutta,heun]]` to compare `rk4` with `midpoint`, `meuler`, and `heun`. This is available when `output=information or plot`. We start with information.

```
> RungeKutta(deq, y(0)=0.5, t=2, submethod=rk4, numsteps=20, digits=10,
output=information, comparewith=[[rungekutta, midpoint], [rungekutta,
meuler], [rungekutta, heun]]);
```

```
[[t, y(t), [R-K 4th Ord.], [Error], [R-K Midpt.], [Error], [R-K Mod. Euler], [Error],
[R-K Heun], [Error]],
```

```
[0., 0.5, 0.5, 0., 0.5, 0., 0.5, 0., 0.5, 0.],
```

```
[0.1000000000, 0.6574145410, 0.6574143750,  $1.660 \cdot 10^{-7}$ , 0.6570000000, 0.0004145410,
0.6572500000, 0.0001645410, 0.6570000000, 0.0004145410],
```

```
[0.2000000000, 0.8292986209, 0.8292982760,  $3.449 \cdot 10^{-7}$ , 0.8284350000, 0.0008636209,
0.8289612500, 0.0003373709, 0.8284350000, 0.0008636209],
```

[0.3000000000, 1.015070596, 1.015070058, $5.38 \cdot 10^{-7}$, 1.013720675, 0.001349921, 1.014552181, 0.000518415, 1.013720675, 0.001349921],

[0.4000000000, 1.214087651, 1.214086906, $7.45 \cdot 10^{-7}$, 1.212211346, 0.001876305, 1.213380160, 0.000707491, 1.212211346, 0.001876305],

[0.5000000000, 1.425639365, 1.425638396, $9.69 \cdot 10^{-7}$, 1.423193537, 0.002445828, 1.424735077, 0.000904288, 1.423193537, 0.002445828],

[0.6000000000, 1.648940600, 1.648939390, 0.000001210, 1.645878859, 0.003061741, 1.647832260, 0.001108340, 1.645878859, 0.003061741],

[0.7000000000, 1.883123646, 1.883122179, 0.000001467, 1.879396139, 0.003727507, 1.881804648, 0.001318998, 1.879396139, 0.003727507],

[0.8000000000, 2.127229536, 2.127227791, 0.000001745, 2.122782733, 0.004446803, 2.125694136, 0.001535400, 2.122782733, 0.004446803],

[0.9000000000, 2.380198444, 2.380196402, 0.000002042, 2.374974920, 0.005223524, 2.378442020, 0.001756424, 2.374974920, 0.005223524],

[1., 2.640859086, 2.640856724, 0.000002362, 2.634797287, 0.006061799, 2.638878432, 0.001980654, 2.634797287, 0.006061799],

[1.100000000, 2.907916988, 2.907914285, 0.000002703, 2.900951002, 0.006965986, 2.905710667, 0.002206321, 2.900951002, 0.006965986],

[1.200000000, 3.179941539, 3.179938470, 0.000003069, 3.172000857, 0.007940682, 3.177510287, 0.002431252, 3.172000857, 0.007940682],

[1.300000000, 3.455351666, 3.455348207, 0.000003459, 3.446360947, 0.008990719, 3.452698867, 0.002652799, 3.446360947, 0.008990719],

[1.400000000, 3.732400017, 3.732396141, 0.000003876, 3.722278847, 0.010121170, 3.729532248, 0.002867769, 3.722278847, 0.010121170],

[1.500000000, 4.009155465, 4.009151145, 0.000004320, 3.997818126, 0.011337339, 4.006083135, 0.003072330, 3.997818126, 0.011337339],

[1.600000000, 4.283483788, 4.283478996, 0.000004792, 4.270839029, 0.012644759, 4.280221864, 0.003261924, 4.270839029, 0.012644759],

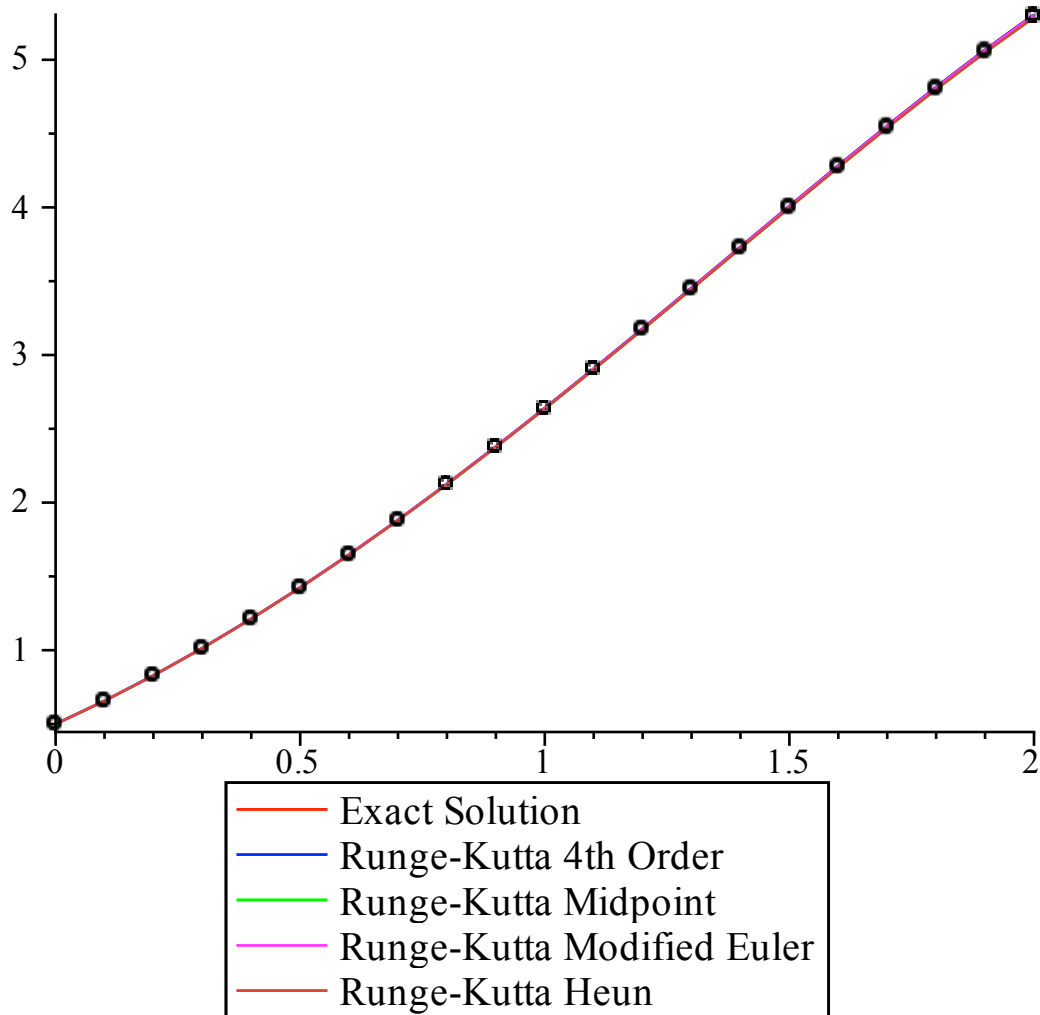
[1.700000000, 4.553026304, 4.553021009, 0.000005295, 4.538977127, 0.014049177, 4.549595159, 0.003431145, 4.538977127, 0.014049177],

[1.800000000, 4.815176268, 4.815170440, 0.000005828, 4.799619725, 0.015556543, 4.811602651, 0.003573617, 4.799619725, 0.015556543],

```
[1.900000000, 5.067052779, 5.067046386, 0.000006393, 5.049879796, 0.017172983,  
5.063370929, 0.003681850, 5.049879796, 0.017172983],  
[2., 5.305471951, 5.305464960, 0.000006991, 5.286567175, 0.018904776, 5.301724877,  
0.003747074, 5.286567175, 0.018904776]]
```

Now we look at **plot**.

```
> RungeKutta(deg, y(0)=0.5, t=2, submethod=rk4, numsteps=20, digits=10,  
output=plot, comparewith=[ [rungekutta, midpoint], [rungekutta, meuler],  
[rungekutta, heun] ] );
```



It is clearly hard to distinguish anything here.