

Adams-Bashforth and Adams-Bashforth-Moulton Methods

The **Adams-Bashforth** method is a multistep method. Only the four-step explicit method is implemented in Maple. It is not clear how the four starting values w_0, \dots, w_3 are obtained, but it doesn't seem to be the Runge-Kutta method of order four as suggested by the text. However, we will compare this method to the Runge-Kutta method of order four, since both have the same order of local truncation error, $O(h^5)$. We examine the method on the IVP

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

```
> restart:with(plots):Digits:=15;
                               Digits := 15
> deq:=D(y)(t)=y(t)-t^2+1;
                               deq := D(y)(t) = y(t) - t^2 + 1
> init:=y(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 the **fourth-order Runge-Kutta** method.

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

Now we apply the **Adams-Bashforth four step method**, which is a [classical](#) method in Maple abbreviated to **adambash**.

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

Now let's compare our results.

```
> printf("      t          y          rk4          |y-rk4|          ab
        |y-ab|\n");
printf("\n");
for i from 0 to 20 do
printf("%4.1f %12.7f %12.7f %12.7f %12.7f %12.7f\n",.1*i,Y(.1*
i),eval(y(t),rk4(.1*i)),abs(Y(.1*i)-eval(y(t),rk4(.1*i))),eval(y
(t),ab(.1*i)),abs(Y(.1*i)-eval(y(t),ab(.1*i))));
od;
t          y          rk4          |y-rk4|          ab          |y-
```

0.0	0.5000000	0.5000000	0.0000000	0.5000000
0.0000000				
0.1	0.6574145	0.6574144	0.0000002	0.6574145
0.0000000				
0.2	0.8292986	0.8292983	0.0000003	0.8292986
0.0000001				
0.3	1.0150706	1.0150701	0.0000005	1.0150701
0.0000005				
0.4	1.2140877	1.2140869	0.0000007	1.2140869
0.0000007				
0.5	1.4256394	1.4256384	0.0000010	1.4256409
0.0000015				
0.6	1.6489406	1.6489394	0.0000012	1.6489452
0.0000046				
0.7	1.8831236	1.8831222	0.0000015	1.8831317
0.0000081				
0.8	2.1272295	2.1272278	0.0000017	2.1272418
0.0000122				
0.9	2.3801984	2.3801964	0.0000020	2.3802157
0.0000172				
1.0	2.6408591	2.6408567	0.0000024	2.6408822
0.0000231				
1.1	2.9079170	2.9079143	0.0000027	2.9079471
0.0000301				
1.2	3.1799415	3.1799385	0.0000031	3.1799798
0.0000382				
1.3	3.4553517	3.4553482	0.0000035	3.4553994
0.0000477				
1.4	3.7324000	3.7323961	0.0000039	3.7324589
0.0000588				
1.5	4.0091555	4.0091511	0.0000043	4.0092272
0.0000717				
1.6	4.2834838	4.2834790	0.0000048	4.2835705
0.0000867				
1.7	4.5530263	4.5530210	0.0000053	4.5531303
0.0001040				
1.8	4.8151763	4.8151704	0.0000058	4.8153003
0.0001240				
1.9	5.0670528	5.0670464	0.0000064	5.0671999
0.0001471				
2.0	5.3054720	5.3054650	0.0000070	5.3056456
0.0001736				

In this case, at least, it appears that the **Runge-Kutta method of order 4** is superior to the **Adams-Bashforth method of four steps**. Let's now use this method as a predictor for the three-step Adams-Moulton method to get an **Adams-Bashforth-Moulton** predictor-corrector method. This is also a **classical** method and is abbreviated as **abmoulton**. The option `corrections` gives the number of times the corrector is applied, the default being 1. Increasing corrections in this case does not help. In any case, it is rarely helpful to have it greater than 4.

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

Now let's compare our results.

```
> printf("  t          y          rk4          |y-rk4|          ab\n");
printf("\n");
for i from 0 to 20 do
```

```

printf("%4.1f %12.7f %12.7f %12.7f %12.7f %12.7f\n",.1*i,Y(.1*
i),eval(y(t),rk4(.1*i)),abs(Y(.1*i)-eval(y(t),rk4(.1*i))),eval(y
(t),abm(.1*i)),abs(Y(.1*i)-eval(y(t),abm(.1*i))));
od;

```

t	y	rk4	y-rk4	ab	y-ab
0.0	0.5000000	0.5000000	0.0000000	0.5000000	0.0000000
0.1	0.6574145	0.6574144	0.0000002	0.6574145	0.0000000
0.2	0.8292986	0.8292983	0.0000003	0.8292986	0.0000001
0.3	1.0150706	1.0150701	0.0000005	1.0150701	0.0000005
0.4	1.2140877	1.2140869	0.0000007	1.2140869	0.0000007
0.5	1.4256394	1.4256384	0.0000010	1.4256385	0.0000009
0.6	1.6489406	1.6489394	0.0000012	1.6489395	0.0000011
0.7	1.8831236	1.8831222	0.0000015	1.8831223	0.0000013
0.8	2.1272295	2.1272278	0.0000017	2.1272280	0.0000016
0.9	2.3801984	2.3801964	0.0000020	2.3801966	0.0000019
1.0	2.6408591	2.6408567	0.0000024	2.6408569	0.0000022
1.1	2.9079170	2.9079143	0.0000027	2.9079144	0.0000026
1.2	3.1799415	3.1799385	0.0000031	3.1799384	0.0000031
1.3	3.4553517	3.4553482	0.0000035	3.4553480	0.0000036
1.4	3.7324000	3.7323961	0.0000039	3.7323958	0.0000042
1.5	4.0091555	4.0091511	0.0000043	4.0091505	0.0000050
1.6	4.2834838	4.2834790	0.0000048	4.2834780	0.0000058
1.7	4.5530263	4.5530210	0.0000053	4.5530196	0.0000067
1.8	4.8151763	4.8151704	0.0000058	4.8151685	0.0000078
1.9	5.0670528	5.0670464	0.0000064	5.0670438	0.0000090
2.0	5.3054720	5.3054650	0.0000070	5.3054616	0.000104

We see a significant improvement from the corrector, but the Runge-Kutta method of order 4 is again superior even here. Of course, all have the same order of truncation error, so this should not be too surprising.

NumericalAnalysis

```
> 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 [AdamsBashforth](#) command, which is short for [InitialValueProblem](#) with **method=adamsbashforth**. 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^2+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: **step2** (local truncation error of $O(h^3)$), **step3** ($O(h^4)$), **step4** ($O(h^5)$), and **step5** ($O(h^6)$). We look at **step4** with a step-size of 0.1 after first resetting the **interface**.

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

10

```
> AdamsBashforth(deq,init,t=2,submethod=step4,numsteps=20,digits=10,
output=information,comparewith=[ [rungekutta,rk4] ] );
```

t	$y(t)$	[A-B 4-Step]	[Error]	[R-K 4th Ord.]	[Error]
0.	0.5	0.5	0.	0.5	0.
0.1000000000	0.6574145410	0.6574143750	1.660 10 ⁻⁷	0.6574143750	1.660 10 ⁻⁷
0.2000000000	0.8292986209	0.8292982760	3.449 10 ⁻⁷	0.8292982760	3.449 10 ⁻⁷
0.3000000000	1.015070596	1.015070058	5.38 10 ⁻⁷	1.015070058	5.38 10 ⁻⁷
0.4000000000	1.214087651	1.214089170	0.000001519	1.214086906	7.45 10 ⁻⁷
0.5000000000	1.425639365	1.425643661	0.000004296	1.425638396	9.69 10 ⁻⁷
0.6000000000	1.648940600	1.648948027	0.000007427	1.648939390	0.000001210
0.7000000000	1.883123646	1.883134837	0.000011191	1.883122179	0.000001467
0.8000000000	2.127229536	2.127245234	0.000015698	2.127227791	0.000001745
0.9000000000	2.380198444	2.380219470	0.000021026	2.380196402	0.000002042
1.000000000	2.640859086	2.640886381	0.000027295	2.640856724	0.000002362
1.100000000	2.907916988	2.907951641	0.000034653	2.907914285	0.000002703
1.200000000	3.179941539	3.179984795	0.000043256	3.179938470	0.000003069
1.300000000	3.455351666	3.455404953	0.000053287	3.455348207	0.000003459
1.400000000	3.732400017	3.732464964	0.000064947	3.732396141	0.000003876
1.500000000	4.009155465	4.009233937	0.000078472	4.009151145	0.000004320
1.600000000	4.283483788	4.283577911	0.000094123	4.283478996	0.000004792
1.700000000	4.553026304	4.553138503	0.000112199	4.553021009	0.000005295
1.800000000	4.815176268	4.815309302	0.000133034	4.815170440	0.000005828
1.900000000	5.067052779	5.067209791	0.000157012	5.067046386	0.000006393
2.000000000	5.305471951	5.305656512	0.000184561	5.305464960	0.000006991

Next we solve the problem by using the [AdamsBashforth](#) command, which is short for [InitialValueProblem](#) with `method=adamsmoulton`. We enter the IVP after resetting some variables that have been given values.

```
> AdamsMoulton(deq,init,t=2,submethod=step3,numsteps=20,digits=10,
output=information,comparewith=[[rungekutta,rk4]]);
```

t	$y(t)$	[A-M 3-Step]	[Error]	[R-K 4th Ord.]	[Error]
0.	0.5	0.5	0.	0.5	0.
0.1000000000	0.6574145410	0.6574143750	1.660 10 ⁻⁷	0.6574143750	1.660 10 ⁻⁷
0.2000000000	0.8292986209	0.8292982760	3.449 10 ⁻⁷	0.8292982760	3.449 10 ⁻⁷
0.3000000000	1.015070596	1.015070050	5.46 10 ⁻⁷	1.015070058	5.38 10 ⁻⁷
0.4000000000	1.214087651	1.214086866	7.85 10 ⁻⁷	1.214086906	7.45 10 ⁻⁷
0.5000000000	1.425639365	1.425638296	0.000001069	1.425638396	9.69 10 ⁻⁷
0.6000000000	1.648940600	1.648939196	0.000001404	1.648939390	0.000001210
0.7000000000	1.883123646	1.883121849	0.000001797	1.883122179	0.000001467
0.8000000000	2.127229536	2.127227278	0.000002258	2.127227791	0.000001745
0.9000000000	2.380198444	2.380195649	0.000002795	2.380196402	0.000002042
1.000000000	2.640859086	2.640855665	0.000003421	2.640856724	0.000002362
1.100000000	2.907916988	2.907912841	0.000004147	2.907914285	0.000002703
1.200000000	3.179941539	3.179936550	0.000004989	3.179938470	0.000003069
1.300000000	3.455351666	3.455345706	0.000005960	3.455348207	0.000003459
1.400000000	3.732400017	3.732392934	0.000007083	3.732396141	0.000003876
1.500000000	4.009155465	4.009147091	0.000008374	4.009151145	0.000004320
1.600000000	4.283483788	4.283473929	0.000009859	4.283478996	0.000004792
1.700000000	4.553026304	4.553014741	0.000011563	4.553021009	0.000005295
1.800000000	4.815176268	4.815162750	0.000013518	4.815170440	0.000005828
1.900000000	5.067052779	5.067037024	0.000015755	5.067046386	0.000006393
2.000000000	5.305471951	5.305453637	0.000018314	5.305464960	0.000006991

Again, rk4 is slightly more accurate than Adams Moulton. Finally, we use the method with a fourth-order Adams Bashforth Predictor and a fourth-order Adams Moulton corrector. One can use step 2,3,4, or 5 predictor correctors, The default is 4.

```
> AdamsBashforthMoulton(deq, init, t=2, submethod=step4, numsteps=20,
  digits=10, output=information, comparewith=[ [rungekutta, rk4] ] );
```

t	$y(t)$	[4th-Ord. A-B-M]	[Error]	[R-K 4th Ord.]	[Error]
0.	0.5	0.5	0.	0.5	0.
0.1000000000	0.6574145410	0.6574143750	$1.660 \cdot 10^{-7}$	0.6574143750	$1.660 \cdot 10^{-7}$
0.2000000000	0.8292986209	0.8292982760	$3.449 \cdot 10^{-7}$	0.8292982760	$3.449 \cdot 10^{-7}$
0.3000000000	1.015070596	1.015070058	$5.38 \cdot 10^{-7}$	1.015070058	$5.38 \cdot 10^{-7}$
0.4000000000	1.214087651	1.214086961	$6.90 \cdot 10^{-7}$	1.214086906	$7.45 \cdot 10^{-7}$
0.5000000000	1.425639365	1.425638498	$8.67 \cdot 10^{-7}$	1.425638396	$9.69 \cdot 10^{-7}$
0.6000000000	1.648940600	1.648939526	0.000001074	1.648939390	0.000001210
0.7000000000	1.883123646	1.883122331	0.000001315	1.883122179	0.000001467
0.8000000000	2.127229536	2.127227941	0.000001595	2.127227791	0.000001745
0.9000000000	2.380198444	2.380196525	0.000001919	2.380196402	0.000002042
1.000000000	2.640859086	2.640856792	0.000002294	2.640856724	0.000002362
1.100000000	2.907916988	2.907914262	0.000002726	2.907914285	0.000002703
1.200000000	3.179941539	3.179938314	0.000003225	3.179938470	0.000003069
1.300000000	3.455351666	3.455347869	0.000003797	3.455348207	0.000003459
1.400000000	3.732400017	3.732395563	0.000004454	3.732396141	0.000003876
1.500000000	4.009155465	4.009150258	0.000005207	4.009151145	0.000004320
1.600000000	4.283483788	4.283477718	0.000006070	4.283478996	0.000004792
1.700000000	4.553026304	4.553019248	0.000007056	4.553021009	0.000005295
1.800000000	4.815176268	4.815168084	0.000008184	4.815170440	0.000005828
1.900000000	5.067052779	5.067043310	0.000009469	5.067046386	0.000006393
2.000000000	5.305471951	5.305461016	0.000010935	5.305464960	0.000006991

We actually lose a touch of accuracy here.