

Stiff Differential Equations

To find methods appropriate to stiff differential equations, we test the methods on the IVP

$$y' = -100y, y(0) = 1.$$

The solution for this equation is

$$y(t) = e^{-100t}.$$

In other words, the solution consists entirely of the transient solution with the steady-state solution being 0. We set Digits very high so as not to run into effects of excessive round-off error.

```
> restart:with(plots):Digits:=1000:
```

```
> deq:=diff(y(t),t)=-100*y(t);
```

$$deq := \frac{d}{dt} y(t) = -100y(t)$$

```
> init:=y(0)=1;
```

$$init := y(0) = 1$$

We compute the exact solution.

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

$$soln := y(t) = e^{-100t}$$

```
> Y:=unapply(rhs(soln),t);
```

$$Y := t \rightarrow e^{-100t}$$

For numerical methods, let's first try Euler's method with $h = .1$.

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

```
eul := proc(x_classical) ... end proc
```

Let's print a table to check our results.

```
> printf("  t      y                      eul                      |y-
eul|\n");
printf("\n");
for i from 0 to 20 do
printf("%4.1f    %16.8e          %16.8e          %12.8e\n", .1*i, Y(.1*i),
eval(y(t),eul(.1*i)),abs(Y(.1*i)-eval(y(t),eul(.1*i))));
od;
```

t	y	eul	y-eul
0.0	1.00000000e+00	1.00000000e+00	0.00000000e+00
0.1	4.53999298e-05	-9.00000000e+00	9.00004540e+00
0.2	2.06115362e-09	8.10000000e+01	8.10000000e+01
0.3	9.35762297e-14	-7.29000000e+02	7.29000000e+02
0.4	4.24835426e-18	6.56100000e+03	6.56100000e+03
0.5	1.92874985e-22	-5.90490000e+04	5.90490000e+04
0.6	8.75651076e-27	5.31441000e+05	5.31441000e+05
0.7	3.97544974e-31	-4.78296900e+06	4.78296900e+06
0.8	1.80485139e-35	4.30467210e+07	4.30467210e+07
0.9	8.19401262e-40	-3.87420489e+08	3.87420489e+08
1.0	3.72007598e-44	3.48678440e+09	3.48678440e+09
1.1	1.68891188e-48	-3.13810596e+10	3.13810596e+10
1.2	7.66764807e-53	2.82429536e+11	2.82429536e+11
1.3	3.48110684e-57	-2.54186583e+12	2.54186583e+12
1.4	1.58042006e-61	2.28767925e+13	2.28767925e+13
1.5	7.17509597e-66	-2.05891132e+14	2.05891132e+14

1.6	3.25748853e-70	1.85302019e+15	1.85302019e+15
1.7	1.47889751e-74	-1.66771817e+16	1.66771817e+16
1.8	6.71418429e-79	1.50094635e+17	1.50094635e+17
1.9	3.04823495e-83	-1.35085172e+18	1.35085172e+18
2.0	1.38389653e-87	1.21576655e+19	1.21576655e+19

It looks like we miss by quite a bit. Obviously, Euler is not for stiff DE's. Let's first try the fourth order Runge-Kutta method with $h = 0.1$.

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

> printf("  t      y                      rk4          |y-
rk4|\n");
printf("\n");
for i from 0 to 20 do
printf("%4.1f    %16.8e    %16.8e    %12.8e\n",.1*i,Y(.1*i),
eval(y(t),rk(.1*i)),abs(Y(.1*i)-eval(y(t),rk(.1*i)))));
od;
```

t	y	rk4	y-rk4
0.0	1.00000000e+00	1.00000000e+00	0.00000000e+00
0.1	4.53999298e-05	2.91000000e+02	2.90999955e+02
0.2	2.06115362e-09	8.46810000e+04	8.46810000e+04
0.3	9.35762297e-14	2.46421710e+07	2.46421710e+07
0.4	4.24835426e-18	7.17087176e+09	7.17087176e+09
0.5	1.92874985e-22	2.08672368e+12	2.08672368e+12
0.6	8.75651076e-27	6.07236592e+14	6.07236592e+14
0.7	3.97544974e-31	1.76705848e+17	1.76705848e+17
0.8	1.80485139e-35	5.14214018e+19	5.14214018e+19
0.9	8.19401262e-40	1.49636279e+22	1.49636279e+22
1.0	3.72007598e-44	4.35441573e+24	4.35441573e+24
1.1	1.68891188e-48	1.26713498e+27	1.26713498e+27
1.2	7.66764807e-53	3.68736278e+29	3.68736278e+29
1.3	3.48110684e-57	1.07302257e+32	1.07302257e+32
1.4	1.58042006e-61	3.12249568e+34	3.12249568e+34
1.5	7.17509597e-66	9.08646242e+36	9.08646242e+36
1.6	3.25748853e-70	2.64416056e+39	2.64416056e+39
1.7	1.47889751e-74	7.69450724e+41	7.69450724e+41
1.8	6.71418429e-79	2.23910161e+44	2.23910161e+44
1.9	3.04823495e-83	6.51578568e+46	6.51578568e+46
2.0	1.38389653e-87	1.89609363e+49	1.89609363e+49

We miss by even more. Let's see if Runge-Kutta-Fehlberg is any better.

```
> rkf:=dsolve({deq,init},y(t), type=numeric,
             method=rkf45, range=0..2);
             rkf:=proc(x_rkf45) ... end proc

> printf("  t      y                      rkf          |y-rkf|
             relative error\n");
printf("\n");
for i from 0 to 20 do
printf("%4.1f    %16.8e    %16.8e    %16.8e    %16.8e\n",.1*i,Y(.1*
i),eval(y(t),rkf(.1*i)),abs(Y(.1*i)-eval(y(t),rkf(.1*i))),abs(Y
(.1*i)-eval(y(t),rkf(.1*i)))/ Y(.1*i));
od;
```

t	y	rkf	y-rkf	relative error
0.0	1.00000000e+00	1.00000000e+00	0.00000000e+00	0.00000000e+00
0.1	4.53999298e-05	4.53811873e-05	1.87424654e-08	

4.12830274e-04			
0.2	2.06115362e-09	8.14057495e-11	1.97974787e-09
9.60504764e-01			
0.3	9.35762297e-14	1.55093615e-08	1.55092679e-08
1.65739397e+05			
0.4	4.24835426e-18	-2.11229871e-09	2.11229871e-09
4.97203996e+08			
0.5	1.92874985e-22	-2.59472032e-08	2.59472032e-08
1.34528608e+14			
0.6	8.75651076e-27	1.03671102e-08	1.03671102e-08
1.18393165e+18			
0.7	3.97544974e-31	1.57381299e-08	1.57381299e-08
3.95883007e+22			
0.8	1.80485139e-35	-8.68889184e-09	8.68889184e-09
4.81418687e+26			
0.9	8.19401262e-40	1.40417802e-08	1.40417802e-08
1.71366348e+31			
1.0	3.72007598e-44	-3.11609774e-09	3.11609774e-09
8.37643576e+34			
1.1	1.68891188e-48	-1.46595566e-08	1.46595566e-08
8.67988248e+39			
1.2	7.66764807e-53	1.38332679e-08	1.38332679e-08
1.80410834e+44			
1.3	3.48110684e-57	-7.50755322e-09	7.50755322e-09
2.15665694e+48			
1.4	1.58042006e-61	-4.55201937e-09	4.55201937e-09
2.88025917e+52			
1.5	7.17509597e-66	1.57694982e-08	1.57694982e-08
2.19781007e+57			
1.6	3.25748853e-70	-1.37012450e-08	1.37012450e-08
4.20607620e+61			
1.7	1.47889751e-74	1.10286861e-09	1.10286861e-09
7.45737014e+64			
1.8	6.71418429e-79	9.10707800e-09	9.10707800e-09
1.35639381e+70			
1.9	3.04823495e-83	-1.63936349e-08	1.63936349e-08
5.37807457e+74			
2.0	1.38389653e-87	9.73841712e-09	9.73841712e-09
7.03695466e+78			

Even though the absolute error is small, the relative error grows exponentially. Also, notice that the numerical solution fluctuates about 0. But the small absolute error doesn't tell us much since the approximated value is basically 0. There is much information to be gained from the two following help pages.

```
> ?dsolve,Stiffness;
```

```
> ?dsolve,Error_Control;
```

The default method for **stiff differential equations** is the [rosenbrock](#) method, invoked by the option **stiff=true**. This option can also be invoked by the option **method=rosenbrock**.

```
> rosen:=dsolve({deq,init},y(t), type=numeric,
  stiff=true,range=0..2);
      rosen := proc(x_rosenbrock) ... end proc
```

```
> printf("    t      y          rosen          |y-rosen|
      relative error\n");
printf("\n");
for i from 0 to 20 do
printf("%4.1f    %16.8e    % 16.8e    %16.8e    %16.8e\n",.1*i,Y(.1*
i),eval(y(t),rosen(.1*i)),abs(Y(.1*i)-eval(y(t),rosen(.1*i))),abs
(Y(.1*i)-eval(y(t),rosen(.1*i)))/ Y(.1*i));
od;
```

t	y	rosen	y-rosen
relative error			
0.0	1.00000000e+00	1.00000000e+00	0.00000000e+00
0.00000000e+00			
0.1	4.53999298e-05	4.54375943e-05	3.76645356e-08
8.29616605e-04			
0.2	2.06115362e-09	-2.82009343e-10	2.34316297e-09
1.13682112e+00			
0.3	9.35762297e-14	-1.11123127e-09	1.11132485e-09
1.18761448e+04			
0.4	4.24835426e-18	6.79635695e-10	6.79635691e-10
1.59976228e+08			
0.5	1.92874985e-22	2.69962494e-10	2.69962494e-10
1.39967603e+12			
0.6	8.75651076e-27	-2.35512590e-10	2.35512590e-10
2.68957118e+16			
0.7	3.97544974e-31	-3.42983510e-10	3.42983510e-10
8.62753983e+20			
0.8	1.80485139e-35	-1.97089927e-10	1.97089927e-10
1.09200086e+25			
0.9	8.19401262e-40	5.75284970e-11	5.75284970e-11
7.02079672e+28			
1.0	3.72007598e-44	2.76232099e-10	2.76232099e-10
7.42544241e+33			
1.1	1.68891188e-48	3.14381218e-10	3.14381218e-10
1.86144240e+38			
1.2	7.66764807e-53	4.10363806e-11	4.10363806e-11
5.35188629e+41			
1.3	3.48110684e-57	1.64396588e-11	1.64396588e-11
4.72253785e+45			
1.4	1.58042006e-61	-2.90015904e-12	2.90015904e-12
1.83505583e+49			
1.5	7.17509597e-66	-1.74353508e-11	1.74353508e-11
2.42998155e+54			
1.6	3.25748853e-70	-2.76181945e-11	2.76181945e-11
8.47837044e+58			
1.7	1.47889751e-74	-3.39009678e-11	3.39009678e-11
2.29231355e+63			
1.8	6.71418429e-79	-3.67359488e-11	3.67359488e-11
5.47139418e+67			
1.9	3.04823495e-83	-3.65754153e-11	3.65754153e-11
1.19988833e+72			
2.0	1.38389653e-87	-3.38716452e-11	3.38716452e-11
2.44755620e+76			

The results are a bit better. But we still have fluctuations about 0. Next, we will use one of the Livermore Stiff ODE Solvers ([lsode](#)). We will choose the **backfull** option, which is a **Gear** method invoking backward differentiation methods.

```
> g:=dsolve({deq,init},y(t), type=numeric,
method=lsode[backfull],start=0);
      g := proc(x_lsode) ... end proc

> printf("    t          y          gear          |y-gear|
relative error\n");
printf("\n");
for i from 0 to 20 do
printf("%4.1f    %16.8e    % 16.8e    %16.8e    %16.8e\n",.1*i,Y(.1*
i),eval(y(t),g(.1*i)),abs(Y(.1*i)-eval(y(t),g(.1*i))),abs(Y(.1*i)
-eval(y(t),g(.1*i)))/ Y(.1*i));
od;
t          y          gear          |y-gear|
```

```

relative error
  0.0      1.00000000e+00      1.00000000e+00      0.00000000e+00
0.00000000e+00
  0.1      4.53999298e-05      4.54654356e-05      6.55058072e-08
1.44286142e-03
  0.2      2.06115362e-09      6.95174554e-09      4.89059192e-09
2.37274499e+00
  0.3      9.35762297e-14      2.16366219e-10      2.16272643e-10
2.31119210e+03
  0.4      4.24835426e-18      1.31856895e-10      1.31856891e-10
3.10371695e+07
  0.5      1.92874985e-22      9.01932456e-11      9.01932456e-11
4.67625419e+11
  0.6      8.75651076e-27      6.57000916e-11      6.57000916e-11
7.50299901e+15
  0.7      3.97544974e-31      4.98084730e-11      4.98084730e-11
1.25290159e+20
  0.8      1.80485139e-35      3.88102128e-11      3.88102128e-11
2.15032734e+24
  0.9      8.19401262e-40      3.08420515e-11      3.08420515e-11
3.76397413e+28
  1.0      3.72007598e-44      2.48691067e-11      2.48691067e-11
6.68510719e+32
  1.1      1.68891188e-48      2.02730870e-11      2.02730870e-11
1.20036381e+37
  1.2      7.66764807e-53      1.66632990e-11      1.66632990e-11
2.17319559e+41
  1.3      3.48110684e-57      1.37815717e-11      1.37815717e-11
3.95896259e+45
  1.4      1.58042006e-61      1.14509179e-11      1.14509179e-11
7.24549010e+49
  1.5      7.17509597e-66      9.54619583e-12      9.54619583e-12
1.33046246e+54
  1.6      3.25748853e-70      7.97654756e-12      7.97654756e-12
2.44868017e+58
  1.7      1.47889751e-74      6.67446932e-12      6.67446932e-12
4.51313853e+62
  1.8      6.71418429e-79      5.58878160e-12      5.58878160e-12
8.32384301e+66
  1.9      3.04823495e-83      4.67997707e-12      4.67997707e-12
1.53530720e+71
  2.0      1.38389653e-87      3.91706508e-12      3.91706508e-12
2.83046095e+75

```

This looks a bit better since the approximated values now march steadily to 0 instead of fluctuating about 0. We try Taylor next.

```

> tay:=dsolve({deq,init},y(t), type=numeric,
method=taylorseries,range=0..2);
      [Length of output exceeds limit of 1000000]
> printf("      t          y          taylor          |y-
taylor|
relative error\n");
printf("\n");
for i from 0 to 20 do
printf("%4.1f      %16.8e      % 16.8e      %16.8e      %16.8e\n",.1*i,Y(.1*
i),eval(y(t),tay(.1*i)),abs(Y(.1*i)-eval(y(t),tay(.1*i))),abs(Y
(.1*i)-eval(y(t),tay(.1*i)))/ Y(.1*i));
od;
      t          y          taylor          |y-taylor|
relative error

```

0.0	1.00000000e+00	1.00000000e+00	0.00000000e+00
0.00000000e+00			
0.1	4.53999298e-05	4.53999298e-05	0.00000000e+00
0.00000000e+00			
0.2	2.06115362e-09	2.06115362e-09	1.32975050e-1001
6.45148661e-993			
0.3	9.35762297e-14	9.35762297e-14	2.01111557e-997
2.14917354e-984			
0.4	4.24835426e-18	4.24835426e-18	1.50832810e-993
3.55038213e-976			
0.5	1.92874985e-22	1.92874985e-22	3.99563305e-989
2.07161808e-967			
0.6	8.75651076e-27	8.75651076e-27	8.26704712e-985
9.44102891e-959			
0.7	3.97544974e-31	3.97544974e-31	5.71606440e-980
1.43784094e-949			
0.8	1.80485139e-35	1.80485139e-35	1.84151842e-975
1.02031582e-940			
0.9	8.19401262e-40	8.19401262e-40	4.86532887e-971
5.93766338e-932			
1.0	3.72007598e-44	3.72007598e-44	1.18518216e-966
3.18590847e-923			
1.1	1.68891188e-48	1.68891188e-48	8.02299625e-963
4.75039364e-915			
1.2	7.66764807e-53	7.66764807e-53	1.35871517e-958
1.77201034e-906			
1.3	3.48110684e-57	3.48110684e-57	1.36211459e-945
3.91287786e-889			
1.4	1.58042006e-61	1.58042006e-61	2.75911702e-897
1.74581245e-836			
1.5	7.17509597e-66	7.17509597e-66	2.58780315e-852
3.60664604e-787			
1.6	3.25748853e-70	3.25748853e-70	3.02987066e-810
9.30124735e-741			
1.7	1.47889751e-74	1.47889751e-74	9.96679552e-771
6.73934163e-697			
1.8	6.71418429e-79	6.71418429e-79	1.80370155e-733
2.68640460e-655			
1.9	3.04823495e-83	3.04823495e-83	3.15293377e-698
1.03434736e-615			
2.0	1.38389653e-87	1.38389653e-87	8.57084745e-665
6.19327189e-578			

[This also works, but is quite slow.