ODE example for MAT360

Andy Long, Spring, 2024

Tea Time Numerical Analysis, Taylor Methods, starting on p. 209

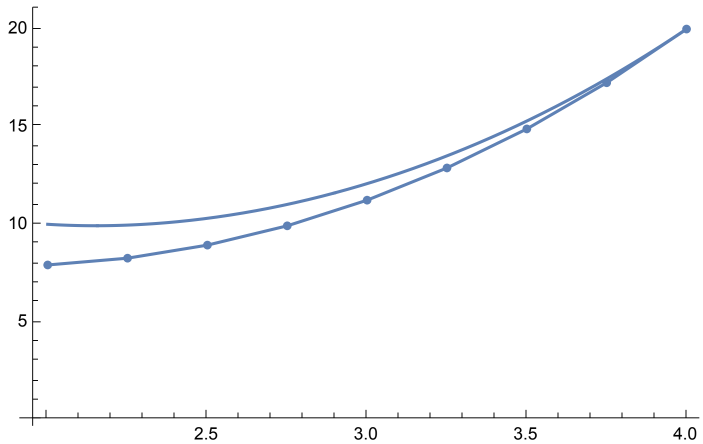
Leave it to Leo to go backwards in time in his first example...:)

In[585]:=
```
euler[f_, y0_, {a_, b_, n_}] :=
 Module[
  (* create some local variables: *)
  {h = N[(b - a) / n], t = a, y = y0, soln},
  (* initialize the solution list *)
  soln = {{t, y}};
  (* iterate, updating y each time: *)
  Do[y = y + h f[t, y];
   (* step time with a constant step-size: *)
   t = t + h;
   (* update soln: *)
   AppendTo[soln, {t, y}
   ],
   (* do this n times: *)
   {n}
  ];
  (* return the solution list of times and y values: *)
  soln
 ]
(* Now we'll recreate Figure 6.2.2, p. 210  *)
f[t_, y_] := -y / t + t^2
y0 = 20;
n = 8;
a = 4;
b = 2;
g[x_] := x^3 / 4 + 16 / x (* exact solution *)

soln = euler[f, y0, {a, b, n}];
MatrixForm[soln]
p0 = ListPlot[soln, Joined → True, InterpolationOrder → 1];
p2 = Plot[g[x], {x, a, b}];
p1 = ListPlot[soln];
Show[p0, p1, p2]
```

Out[593]//MatrixForm=

4	20
3.75	17.25
3.5	14.884375
3.25	12.8850446428571
3.	11.2355769230769
2.75	9.921875
2.5	8.93323863636364
2.25	8.2640625
2.	7.91666666666667

Out[597]=

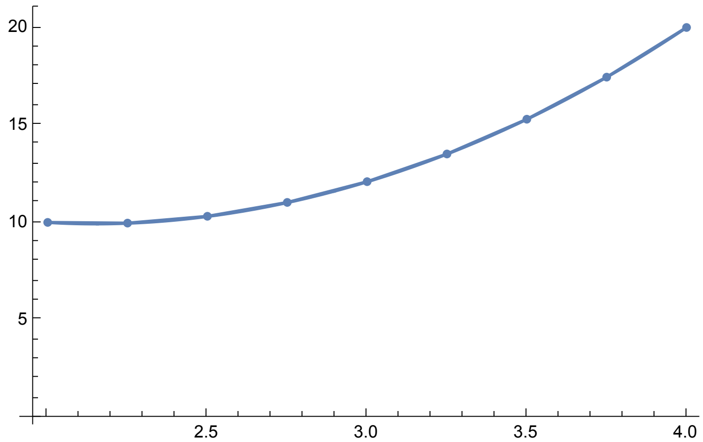


In[598]:=
```
taylor2[f_, fp_, y0_, {a_, b_, n_}] :=
 Module[
  (* create some local variables: *)
  {h = N[(b - a) / n], t = a, y = y0, soln},
  (* initialize the solution list *)
  soln = {{t, y}};
  (* iterate, updating y each time: *)
  Do[y = y + h ( f[t, y] + h / 2 * fp[t, y]);
   (* step time with a constant step-size: *)
   t = t + h;
   (* update soln: *)
   AppendTo[soln, {t, y}
   ],
   (* do this n times: *)
   {n}
  ];
  (* return the solution list of times and y values: *)
  soln
 ]

f[t_, y_] := -y / t + t^2
(* This calculation is done on page 209 *)
fp[t_, y_] := 2 y / t^2 + t
y0 = 20;
n = 8;
a = 4;
b = 2;
g[x_] := x^3 / 4 + 16 / x (* exact solution *)

soln = taylor2[f, fp, y0, {a, b, n}];
MatrixForm[soln]
p0 = ListPlot[soln, Joined → True, InterpolationOrder → 1];
p2 = Plot[g[x], {x, a, b}];
p1 = ListPlot[soln];
Show[p0, p1, p2]
```

Out[607]//MatrixForm=

4	20
3.75	17.453125
3.5	15.2957986111111
3.25	13.5132704435941
3.	12.0936504655486
2.75	11.0291883547996
2.5	10.3183046585813
2.25	9.96894317102529
2.	10.0043643958432

Out[611]=

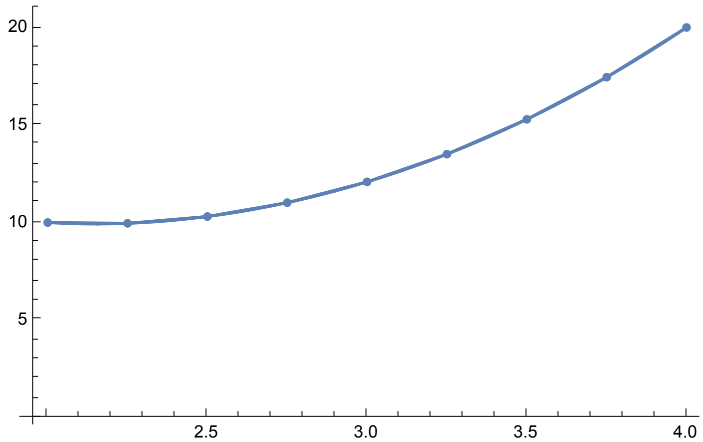


In[612]:=
```
taylor3[f_, fp_, fpp_, y0_, {a_, b_, n_}] :=
 Module[
  (* create some local variables: *)
  {h = N[(b - a) / n], t = a, y = y0, soln},
  (* initialize the solution list *)
  soln = {{t, y}};
  (* iterate, updating y each time: *)
  Do[y = y + h ( f[t, y] + h / 2 * (fp[t, y] + h / 3 fpp[t, y]));
   (* step time with a constant step-size: *)
   t = t + h;
   (* update soln: *)
   AppendTo[soln, {t, y}
   ],
   (* do this n times: *)
   {n}
  ];
  (* return the solution list of times and y values: *)
  soln
 ]
(* Now we'll recreate Table 6.2, p. 212 *)
f[t_, y_] := -y / t + t^2
fp[t_, y_] := 2 y / t^2 + t
(*  I did the following calculation, and hope that it's right...:) *)
fpp[t_, y_] := 2 (t^3 - 3 y) / t^3 + 1
y0 = 20;
n = 8;
a = 4;
b = 2;
g[x_] := x^3 / 4 + 16 / x (* exact solution *)

soln = taylor3[f, fp, fpp, y0, {a, b, n}];
MatrixForm[soln]
p0 = ListPlot[soln, Joined → True, InterpolationOrder → 1];
p2 = Plot[g[x], {x, a, b}];
p1 = ListPlot[soln];
Show[p0, p1, p2]
```

Out[622]//MatrixForm=

4	20
3.75	17.4501953125
3.5	15.2900185185185
3.25	13.5048076611597
3.	12.08282105533
2.75	11.0165611627877
2.5	10.3048895885208
2.25	9.9565448328466
2.	9.99628071047217

Out[626]=



#1-3, ac

In[627]:=
```
f[t_, y_] := 3 t - 2 y
fp[t_, y_] := 3 - 2 (3 t - 2 y)
(*  I did the following calculation, and hope that it's right...:)
     Notice that I cheat and don't simplify y', leaving it as f[t,y]. *)
fpp[t_, y_] := -6 + 4 f[t, y]
y0 = 1;
n = 2;
a = 1;
b = 2;
soln = euler[f, y0, {a, b, n}];
MatrixForm[soln]
soln = taylor2[f, fp, y0, {a, b, n}];
MatrixForm[soln]
soln = taylor3[f, fp, fpp, y0, {a, b, n}];
MatrixForm[soln]
```

Out[635]//MatrixForm=

$$\begin{pmatrix} 1 & 1 \\ 1.5 & 1.5 \\ 2. & 2.25 \end{pmatrix}$$

Out[637]//MatrixForm=

$$\begin{pmatrix} 1 & 1 \\ 1.5 & 1.625 \\ 2. & 2.3125 \end{pmatrix}$$

Out[639]//MatrixForm=

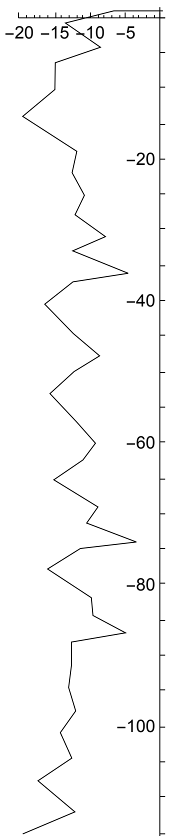
$$\begin{pmatrix} 1 & 1 \\ 1.5 & 1.58333333333333 \\ 2. & 2.27777777777778 \end{pmatrix}$$

It’s easy to operate on systems:

Here’s the pendulum problem:

In[640]:=
```
Clear[x, y, z, f, c, m, g, l]
(* We're thinking of f as a vector-valued function,
obtained by reducing the second order equation for the pendulum to a
  first order system. The arguments a time t and a location (u,v): *)
f[t_, {u_, v_}] := {
  -c / m u - g / l Sin[v],
  u
 }
g = 9.81; (* m/s2 *)
c = 1.1;
l = .31; (* m *)
m = 70; (* kg *)
theta0 = Pi / 3;
thetaprime0 = 0;

(* We just give an initial value that is also a list, of the proper size: *)
soln = euler[f, {thetaprime0, theta0}, {0, 10, 40}];
curve = Table[soln〚k〛〚2〛, {k, 1, Length[soln]}];
Show[Graphics[{Line[curve]}], Axes → Automatic]
```

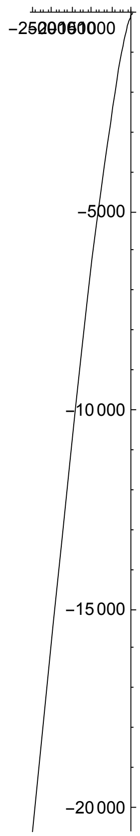
Out[647]=



Let’s try that again, with Taylor 3:

In[648]:=
```
soln = taylor3[f, fp, fpp, {thetaprime0, theta0}, {0, 10, 40}];
curve = Table[soln⟦k⟧⟦2⟧, {k, 1, Length[soln]}];
Show[Graphics[{Line[curve]}], Axes → Automatic]
```

Out[650]=

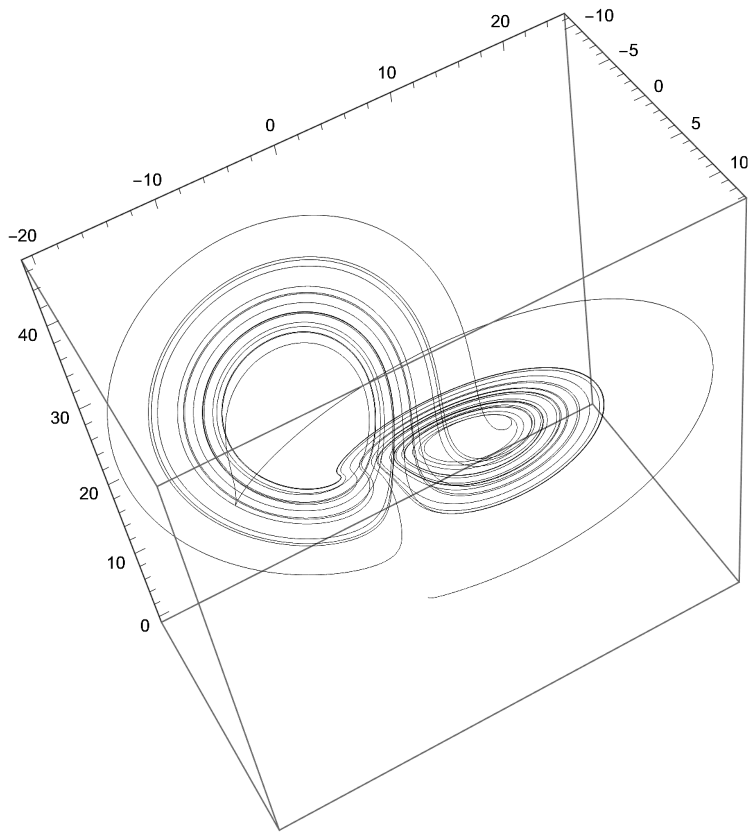


In[651]:=
```
Clear[g]
```

Here is the lovely Lorenz System, exhibiting chaos:

In[652]:=
```
Clear[x, y, z, f]
f[t_, {x_, y_, z_}] := {
  -3. (x - y),
  -x z + 26.5 x - y,
  x y - z
 }
soln = euler[f, {0.0, 1.0, 0.0}, {0, 50, 40 000}];
curve = Table[soln⟦k⟧⟦2⟧, {k, 1, Length[soln]}];
Show[Graphics3D[{Line[curve]}], Axes → Automatic]
```

Out[656]=



In[657]:=

In[658]:=

```
soln = taylor3[f, fp, fpp, {0.0, 1.0, 0.0}, {0, 50, 40000}];
curve = Table[soln[[k]][[2]], {k, 1, Length[soln]}];
Show[Graphics3D[{Line[curve]}], Axes → Automatic]
```

Out[660]=