1 Taylor methods

Context: We have an initial value problem
\[
\begin{aligned}
  \frac{dy}{dt} &= f(t, y(t)) \\
y(t_0) &= y_0.
\end{aligned}
\] (1)

Recall the Taylor expansion for a sufficiently nice function \( y \) of \( t \),
\[
y(t_{i+1}) = y(t_i) + h y'(t_i) + \frac{h^2}{2!} y''(t_i) + \cdots + \frac{h^n}{n!} y^{(n)}(t_i) + \frac{h^{n+1}}{(n+1)!} y^{(n+1)}(\xi_i) \] (2)
where \( h = t_{i+1} - t_i \) and \( \xi \in (t_i, t_{i+1}) \). By letting \( n = 1 \) we get
\[
y(t_{i+1}) = y(t_i) + h y'(t_i) + \frac{h^2}{2!} y''(t_i) \]
which upon truncation of the term \( E_1 := (h^2/2)y''(\xi_i) \) yields Euler’s method
\[
y_{i+1} = y_i + h y'_i.\] (3)

Recalling the context, \( y' = f(t, y(t)) \) this is
\[
y_{i+1} = y_i + h f(t_i, y_i).\]

Since \( E_1 \) is a multiple of \( h^2 \), we say the local error of Euler’s method is on the order of \( h^2 \). We also write \( O(h^2) \). If we apply Euler’s method over an interval \([a, b]\) with \( N \) iterations so that \( h = (b - a)/N \), then the cumulative error is roughly \( 1/h \) times the local error, i.e. the cumulative error is \( O(h) \).

It is tempting to improve our accuracy by letting \( n = 2 \)—in this case \( E_2 = (h^3/3!)y'''(\xi) \) hence the cumulative error is of the order \( h^2 \). Let us pursue this briefly.
\[
y(t_{i+1}) = y(t_i) + h y'(t_i) + \frac{h^2}{2} y''(t_i) + \frac{h^3}{3!} y'''(\xi_i) \] (4)
for at least one \( \xi \in (t_i, t_{i+1}) \). In order for the formula to be useful, we need to rewrite (4) in terms of the right hand side of the DE, i.e. \( y'(t) = f(t, y(t)) \). Here is where things get a bit more difficult than in the procedure for Euler’s method. We must also replace \( y'' \) in (4) with an expression in terms of \( f \),
\[
\frac{d}{dt} y' = \frac{d}{dt} f(t, y(t)) = \frac{\partial f}{\partial t}(t, y(t)) + \frac{\partial f}{\partial y}(t, y(t)) f(t, y(t)).
\]
Substituting this term back into (4) would yield the Taylor method of order 2. It is somewhat lacking in appeal since we will always have to calculate the \( t \) and \( y \) partials of \( f \) and this could be difficult in some cases.
2 Runge-Kutta

Our goal is to get a method of higher accuracy that doesn’t require that we compute partials of $f$ from (1). If we write down the first few terms of the multivariable Taylor formula

$$f(t + \alpha, y + \beta) \approx f(t, y) + \left[ \alpha \frac{\partial f}{\partial t}(t, y) + \beta \frac{\partial f}{\partial y} f(t, y) \right]$$

and then if we cleverly let $\alpha = h/2$ and $\beta = (h/2)f(t, y(t))$, we can use this to make a more convenient replacement of $(h/2)y''$ in (4), namely

$$(h/2)y'' = f \left[ t + h/2, y(t) + (h/2)f(t, y(t)) \right] - f (t, y(t))$$

which after cancellation of $hf(t, y(t))$ terms yields

$$y(t_{i+1}) = y(t_i) + hf \left[ t_i + h/2, y(t_i) + (h/2)f(t_i, y(t_i)) \right] + O(h^3).$$

The algorithm we get is known as the RK2 midpoint method,

$$y_{i+1} = y_i + hf \left[ t_i + h/2, y_i + (h/2)f(t_i, y_i) \right]$$

The algorithm is globally $O(h^2)$ like the Taylor method of order 2 but it doesn’t require computation of the partials of $f$. 
3 Implicit methods: Backwards Euler

The backwards Euler method is a variation of (3),

\[ y_{i+1} = y_i + hf(t_{i+1}, y_{i+1}). \]  

(9)

It is called an implicit method because evaluation of \( f(t_{i+1}, y_{i+1}) \) depends on knowing the future value: \( y_{i+1} \). If \( f \) is linear, then this is not a problem. If \( f \) is nonlinear, then a common strategy is to apply Newton’s method.

Although implicit methods present computational difficulties, they often provide other benefits. Consider the following simple example: \( y' = \lambda y \). Euler’s method yields \( y_1 = y_0 + h\lambda y_0 = (1 + h\lambda)y_0 \) and thus \( y_n = (1 + h\lambda)^n y_0 \). Whereas for the backwards Euler method \( y_1 = y_0 + h\lambda y_1 \) and solving for \( y_1 \), we obtain \( (1 - h\lambda)y_1 = y_0 \). In general then, \( y_n = (1 - h\lambda)^{-n} y_0 \).

We know the solution to \( y' = \lambda y \) is \( y(t) = e^{\lambda t} \). If \( \lambda < 0 \), then the solution is bounded for \( t > 0 \). Euler’s method will be bounded (and hence consistent with the true solution) if and only if \(-1 < 1 + h\lambda < 1\). Given that \( h\lambda < 0 \), the lefthand inequality \(-1 < 1 + h\lambda \) tells the tale: it is required that \( h < 2/|\lambda| \). Euler’s method is said to be conditionally stable. On the other hand, the backwards Euler method demonstrates boundedness irrespective of \( h \) and so this method is unconditionally stable. Better! To learn more about this situation, look up stiff differential equations in almost any numerical methods text.

As mentioned above, for nonlinear \( f \) one often employs Newton’s method. Given \( t_{i+1} \) and \( y_i \), we seek \( y_{i+1} \) with the property that \( y_{i+1} = y_i + hf(t_{i+1}, y_{i+1}) \). If we set \( F(t_{i+1}, y) := y_{i+1} - (y_i + hf(t_{i+1}, y)) \) then we seek a \( y \) that makes \( F(t_{i+1}, y) = 0 \). Newton’s method then instructs us that we should iterate

\[ w_{j+1} = w_j - F(t_{i+1}, w_j) \frac{\partial F}{\partial y}(t_{i+1}, w_j) \]

When we are satisfied with the result, we have our value for \( y_{i+1} \).
Exercise: Our text cites the algorithm for RK4 for a system in exercise 21. Let’s be different. Consider the system,

\[
\begin{aligned}
    x' &= f(t, x, y) \\
y' &= g(t, x, y).
\end{aligned}
\]  

(10)

then the two equation version of the RK2 midpoint method is

\[
\begin{aligned}
x_{i+1} &= x_i + hf \left[ t_i + h/2, x_i + (h/2)f(t_i, x_i, y_i), y_i + (h/2)g(t_i, x_i, y_i) \right] \\
y_{i+1} &= y_i + hg \left[ t_i + h/2, x_i + (h/2)f(t_i, x_i, y_i), y_i + (h/2)g(t_i, x_i, y_i) \right]
\end{aligned}
\]  

(11)

Do problem 21 in the text but implement RK2 midpoint and not RK4. Let me know how it goes. I haven’t tried this yet.

Exercise: The Maple code below almost implements the scheme outlined above for \( f(t, y) = 5e^{5t}(y - t)^2 + 1 \). Finish the program and integrate \( y' = f(t, y) \) on the interval \([0, 1]\) with \( h = 0.25 \). Compare your results to the true values. Note: in this context, integrate means use the numerical quadrature method RK2 midpoint.

```maple
> restart;
> soln:=t->t-exp(-5*t);
> f:=(t,y)->5*exp(5*t)*(y-t)^2+1;
> D2f:=D[2](f);
> w0:=-1;h:=0.25;t:=0.0;
> w:=w0;
> w:=w-(w-w0-h*f(t+h,w))/(1-h*D2f(t+h,w));soln(t+h);
> t:=t+h;w0:=w;
```