```
> restart;
```

Runge-Kutta stencils to solve $\frac{d}{dx}y(x) = f(x, y(x))$

The purpose of this worksheet is to introduce and derive the Runge-Kutta stencils to solve the first order equation $\frac{d}{dx}y(x) = f(x, y(x))$, and study some of their properties.

General form of the stencils

The generic M-stage Runge-Kutta stencil to solve $\frac{d}{dx}y(x) = f(x, y(x))$ involves defining M quantities k_n as follows:

$$k_n = f\left(x_i + c_n h, y_i + h\sum_{m=1}^{M} \alpha_{n,m} k_m\right)$$

In this expression, $y_i \approx y(x_i)$ represents our numeric approximation to solution of the differential equation at $x = x_i$ as usual. The coefficients c_n and $\alpha_{n,m}$ are constants that we are going to choose shortly, and $h = x_{i+1} - x_i$ is the stepsize. Once one calculates k_n given (x_i, y_i), the value of $y_{i+1} \approx y(x_{i+1})$ is given by

$$y_{i+1} = y_i + h\sum_{n=1}^{M} b_n k_n.$$

In this expression, the b_n are constants. Here is what these equations look like for a particular choice of M (which you can change):

```
> unassign(alpha,b,c):

  M := 4;
  for n from 1 to M do:
     k_def[n] := k[n] = f(x[i]+c[n]*h,y[i]+h*add(alpha[n,m]*k[m],m=1..M)):
  od;

  evolve := y[i+1] = y[i] + h*add(b[n]*k[n],n=1..M);
```

$$M := 4$$

$$k\_def_1 := k_1 = f\left(x_i + c_1\, h, y_i + h\left(\alpha_{1,1}\, k_1 + \alpha_{1,2}\, k_2 + \alpha_{1,3}\, k_3 + \alpha_{1,4}\, k_4\right)\right)$$

$$k\_def_2 := k_2 = f\left(x_i + c_2\, h, y_i + h\left(\alpha_{2,1}\, k_1 + \alpha_{2,2}\, k_2 + \alpha_{2,3}\, k_3 + \alpha_{2,4}\, k_4\right)\right)$$

$$k\_def_3 := k_3 = f\left(x_i + c_3\, h, y_i + h\left(\alpha_{3,1}\, k_1 + \alpha_{3,2}\, k_2 + \alpha_{3,3}\, k_3 + \alpha_{3,4}\, k_4\right)\right)$$

$$k\_def_4 := k_4 = f\left(x_i + c_4\, h, y_i + h\left(\alpha_{4,1}\, k_1 + \alpha_{4,2}\, k_2 + \alpha_{4,3}\, k_3 + \alpha_{4,4}\, k_4\right)\right)$$

$$evolve := y_{i+1} = y_i + h\left(b_1\, k_1 + b_2\, k_2 + b_3\, k_3 + b_4\, k_4\right) \qquad \textbf{(1.1}$$

A given Runge-Kutta scheme is defined when we assign definate values to the constants $\left\{\alpha_{n,m}, b_n, c_n\right\}_{n,m=1}^{M}$. In the literature, these are often presented as a Butcher tableau, which is a matrix arranged in the following way:

```
> A := Matrix([seq([seq(alpha[n,m],m=1..M)],n=1..M)]):
  B := Vector[row]([seq(b[n],n=1..M)]):
  C := Vector[column]([seq(c[n],n=1..M)]):
  Butcher := Matrix([[C,A],[``,B]]);
```

$$Butcher := \begin{bmatrix} c_1 & \alpha_{1,1} & \alpha_{1,2} & \alpha_{1,3} & \alpha_{1,4} \\ c_2 & \alpha_{2,1} & \alpha_{2,2} & \alpha_{2,3} & \alpha_{2,4} \\ c_3 & \alpha_{3,1} & \alpha_{3,2} & \alpha_{3,3} & \alpha_{3,4} \\ c_4 & \alpha_{4,1} & \alpha_{4,2} & \alpha_{4,3} & \alpha_{4,4} \\ & b_1 & b_2 & b_3 & b_4 \end{bmatrix} \qquad \textbf{(1.2}$$

You might notice an immediate barrier to implementing this scheme: the equations **(1.1)** actually involve the k_n's on both the lefthand and righthand sides. Hence, it will not be in general possible to analytically solve for the k_n's except for very specific forms of $f(x, y)$. In this sense, the most general Runge-Kutta methods are implicit schemes: the k_n's are defined implicitly. However, we can obtain an explicit scheme by setting a number of the $\alpha_{n,m}$ coefficients equal to zero:

```
> for n from 1 to M do:
     for m from n to M do:
        alpha[n,m] := 0:
     od;
  od;
```

```
Butcher;
```

$$\begin{bmatrix} c_1 & 0 & 0 & 0 & 0 \\ c_2 & \alpha_{2,1} & 0 & 0 & 0 \\ c_3 & \alpha_{3,1} & \alpha_{3,2} & 0 & 0 \\ c_4 & \alpha_{4,1} & \alpha_{4,2} & \alpha_{4,3} & 0 \\ & b_1 & b_2 & b_3 & b_4 \end{bmatrix} \tag{1.3}$$

With these choices, we see that the k_n's are now defined explicitly:

```
> for n from 1 to M do;
      k_def[n];
  od;
```

$$k_1 = f\left(x_i + c_1\, h, y_i\right)$$

$$k_2 = f\left(x_i + c_2\, h, y_i + h\, \alpha_{2,1}\, k_1\right)$$

$$k_3 = f\left(x_i + c_3\, h, y_i + h\, \left(\alpha_{3,1}\, k_1 + \alpha_{3,2}\, k_2\right)\right)$$

$$k_4 = f\left(x_i + c_4\, h, y_i + h\, \left(\alpha_{4,1}\, k_1 + \alpha_{4,2}\, k_2 + \alpha_{4,3}\, k_3\right)\right) \tag{1.4}$$

That is, the first equation allows you to calculate k_1, which can then be used in the second to give you k_2, etc. This is the general form of an explicit Runge-Kutta M-stage stencil. It is traditional (but not necessary) to also set $c_1 = 0$, resulting in the following Butcher tableau:

```
> c[1] := 0;
  Butcher, map(x->if (x=0) then `` else x end if,Butcher);
```

$$c_1 := 0$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ c_2 & \alpha_{2,1} & 0 & 0 & 0 \\ c_3 & \alpha_{3,1} & \alpha_{3,2} & 0 & 0 \\ c_4 & \alpha_{4,1} & \alpha_{4,2} & \alpha_{4,3} & 0 \\ & b_1 & b_2 & b_3 & b_4 \end{bmatrix}, \begin{bmatrix} & & & & \\ c_2 & \alpha_{2,1} & & & \\ c_3 & \alpha_{3,1} & \alpha_{3,2} & & \\ c_4 & \alpha_{4,1} & \alpha_{4,2} & \alpha_{4,3} & \\ & b_1 & b_2 & b_3 & b_4 \end{bmatrix} \tag{1.5}$$

Notice that we have written the tableau in two ways, the second just has blanks where all the zeros reside and is how the tableau is usually written in books. Now, how do we choose the remaining coefficients in the Butcher tableau? The idea is to select them in such a way that the local error in the stencil is $\mathrm{O}\left(h^{L+1}\right)$ where L is the desired global error in the solution; that is

$$\left(y(x+h) - y(x) - h \sum_{n=1}^{M} b_n k_n \Big|_{(x_i, y_i) = (x, y(x))} \right) = \mathrm{O}\left(h^{L+1}\right).$$

More concretely, to determine the coefficients we need to:

1. Construct an expression for the local error by first subtracting the RHS from the LHS of **evolve**, subbing in the k_n definitions, and then making the substitutions $\{x_i, y_i, y_{i+1}\} \mapsto \{x, y(x), y(x+h)\}$
2. Expand the result in a Taylor series in h and simplify derivatives using the differential equation $y'(x) = f(x, y(x))$.
3. Select $\{\alpha_{n,m}, b_n, c_n\}$ in such a way that the coefficients of all terms $\propto h^p$ are zero ($p \leq L$). Note that our choices have to work ***for all*** possible forms of $f(x, y)$.

Obviously, we want to choose L as big as possible in order minimize the one-step error. It turns out for $M \leq 4$ there are enough coefficents in the Butcher tableau to achieve $M = L$, but for higher M the global error cannot be made that small. In the following sections, we will construct explicit Runge-Kutta stencils for $M = 2$ and $M = 4$ and hence choose $M = L$.

The $M = 2$ Huen (modified Euler) method

```
> restart;
```

In this section, we will construct an example of an $M = 2$ explicit Runge-Kutta stencil. The relevant equations are (see previous section):

```
> M := 2;

  for n from 1 to M do:
     for m from n to M do:
        alpha[n,m] := 0:
     od;
  od;

  c[1] := 0;

  for n from 1 to M do:
```

```
    k_def[n] := k[n] = f(x[i]+c[n]*h,y[i]+h*add(alpha[n,m]*k[m],m=1..M)):
od;

evolve := y[i+1] = y[i] + h*add(b[n]*k[n],n=1..M);

A := Matrix([seq([seq(alpha[n,m],m=1..M)],n=1..M)]):
B := Vector[row]([seq(b[n],n=1..M)]):
C := Vector[column]([seq(c[n],n=1..M)]):
Butcher := map(x->if (x=0) then `` else x end if,Matrix([[C,A],[``,B]]));
```

$$M := 2$$

$$c_1 := 0$$

$$k\_def_1 := k_1 = f\left(x_i, y_i\right)$$

$$k\_def_2 := k_2 = f\left(x_i + c_2\, h, y_i + h\, \alpha_{2,1}\, k_1\right)$$

$$evolve := y_{i+1} = y_i + h\left(b_1\, k_1 + b_2\, k_2\right)$$

$$Butcher := \begin{bmatrix} & & \\ c_2 & \alpha_{2,1} & \\ & b_1 & b_2 \end{bmatrix} \qquad \textbf{(2.1}$$

It will be useful to have a set indicating what we want to solve for:

```
> unknowns := {seq(seq(alpha[n,m],m=1..n-1),n=2..M),seq(b[n],n=1..M),seq(c[n],n=2..M)};
```

$$unknowns := \left\{\alpha_{2,1}, b_1, b_2, c_2\right\} \qquad \textbf{(2.2}$$

We could have also obtained this by using the **indets** function to find all the indeterminant quantities (all symbols that have not been assigned a value) in the Butcher tableau:

```
> unknowns := indets(Butcher);
```

$$unknowns := \left\{, \alpha_{2,1}, b_1, b_2, c_2\right\} \qquad \textbf{(2.3}$$

Actually, this returns the set we want with an extra element `` corresponding to all the blank elements in the Butcher tableau. This can be removed using the **minus** operator:

```
> unknowns := unknowns minus {``};
```

$$unknowns := \left\{\alpha_{2,1}, b_1, b_2, c_2\right\} \qquad \textbf{(2.4}$$

We now need to construct an expression for the local error. First, we form the LHS-RHS of **evolve**:

```
> Error := (lhs-rhs)(evolve);
```

$$Error := y_{i+1} - y_i - h\left(b_1 k_1 + b_2 k_2\right) \tag{2.5}$$

Then, let us substitute in the k_n definitions (we convert `k_def` to a list first so we can put them all in at once:

```
> Error := subs(convert(k_def,list),Error);
```

$$Error := y_{i+1} - y_i - h\left(b_1 f\left(x_i, y_i\right) + b_2 f\left(x_i + c_2\, h, y_i + h\,\alpha_{2,1}\, k_1\right)\right) \tag{2.6}$$

Actually, we see in this expression that there is still a k_1 (due to the fact that the definition of k_2 has k_1 in it), so we need to do this again:

```
> Error := subs(convert(k_def,list),Error);
```

$$Error := y_{i+1} - y_i - h\left(b_1 f\left(x_i, y_i\right) + b_2 f\left(x_i + c_2\, h, y_i + h\,\alpha_{2,1} f\left(x_i, y_i\right)\right)\right) \tag{2.7}$$

This expression has no reference to the k_n's. Note we could have generated the error in a single line by recursively subbing in `k_def` using the `$` operator. It is not hard to convince yourself that the number of times `k_def` needs to be subsituted in for an M-stage method is M, so we take that into account in our code:

```
> Error := subs(convert(k_def,list)$M,(lhs-rhs)(evolve));
```

$$Error := y_{i+1} - y_i - h\left(b_1 f\left(x_i, y_i\right) + b_2 f\left(x_i + c_2\, h, y_i + h\,\alpha_{2,1} f\left(x_i, y_i\right)\right)\right) \tag{2.8}$$

Now, we need to replace the numeric data with the exact function values they are meant to represent:

```
> Subs := {y[i] = y(x),y[i+1] = y(x+h), x[i] = x};
  Error := subs(Subs,Error);
```

$$Subs := \left\{x_i = x, y_i = y(x), y_{i+1} = y(x+h)\right\}$$

$$Error := y(x+h) - y(x) - h\left(b_1 f(x, y(x)) + b_2 f\left(x + c_2\, h, y(x) + h\,\alpha_{2,1} f(x, y(x))\right)\right) \tag{2.9}$$

We now expand the Error in a Taylor series of order M:

```
> Error := series(Error,h,M+1);
```

$$Error := \left(\mathrm{D}(y)(x) - b_1 f(x, y(x)) - b_2 f(x, y(x))\right) h + \left(\frac{1}{2}\,\mathrm{D}^{(2)}(y)(x) - b_2\left(\mathrm{D}_1(f)\,(x, y(x))\, c_2 + \mathrm{D}_2(f)\,(x,\right.\right. \tag{2.10}$$

$$\left.\left. y(x))\,\alpha_{2,1} f(x, y(x))\right)\right) h^2 + \mathrm{O}\left(h^3\right)$$

We have not yet used the fact that $y(x)$ is the solution to $y'(x) = f(x, y(x))$. We can use this to remove the derivatives of $y(x)$ appearing above:

```
> derivative[1] := D(y)(x) = f(x,y(x));
  for j from 2 to M do:
       derivative[j] := subs(derivative[1],convert(diff(derivative[j-1],x),D)):
  od;
```

```
Error := subs(convert(derivative,list),Error);
```

$$derivative_1 := \mathrm{D}(y)(x) = f(x, y(x))$$

$$derivative_2 := \mathrm{D}^{(2)}(y)(x) = \mathrm{D}_1(f)(x, y(x)) + \mathrm{D}_2(f)(x, y(x)) f(x, y(x))$$

$$Error := \left(f(x, y(x)) - b_1 f(x, y(x)) - b_2 f(x, y(x))\right) h + \left(\frac{1}{2}\mathrm{D}_1(f)(x, y(x)) + \frac{1}{2}\mathrm{D}_2(f)(x, y(x)) f(x, y(x)) - b_2\left(\mathrm{D}_1(f)(x, y(x))\, c_2 + \mathrm{D}_2(f)(x, y(x))\, \alpha_{2,1} f(x, y(x))\right)\right) h^2 + \mathrm{O}(h^3) \quad \textbf{(2.11}$$

Now, to determine what conditions the coefficients need to satisfy for the error to be $\mathrm{O}(h^3)$, we can rearrange this expression to be a polynomial in $\left\{h, f, \frac{\partial}{\partial x} f, \frac{\partial}{\partial y} f, \mathrm{O}(h^3)\right\}$. This can be accomplishs using the **collect** command, but we first need to write down a set corresponding to the terms we want to collect together. To do this, first lets get a set with all the indeterminants in **Error**:

```
> vars := indets(Error);
```

$$vars := \left\{h, x, \alpha_{2,1}, b_1, b_2, c_2, f(x, y(x)), y(x), \mathrm{D}_1(f)(x, y(x)), \mathrm{D}_2(f)(x, y(x))\right\} \quad \textbf{(2.12}$$

Now, we subtract out the unknown coefficients:

```
> vars := vars minus unknowns;
```

$$vars := \left\{h, x, f(x, y(x)), y(x), \mathrm{D}_1(f)(x, y(x)), \mathrm{D}_2(f)(x, y(x))\right\} \quad \textbf{(2.13}$$

Finally, there is no need to factor over x or $y(x)$, so we take these out, and the **indets** function does not count the $\mathrm{O}(h^3)$ term as an indeterminant, so we add this back in:

```
> vars := vars minus {x,y(x)};
  vars := vars union {O(h^(M+1))};
```

$$vars := \left\{h, f(x, y(x)), \mathrm{D}_1(f)(x, y(x)), \mathrm{D}_2(f)(x, y(x))\right\}$$

$$vars := \left\{h, \mathrm{O}(h^3), f(x, y(x)), \mathrm{D}_1(f)(x, y(x)), \mathrm{D}_2(f)(x, y(x))\right\} \quad \textbf{(2.14}$$

Now we can rewrite **Error** as a polynomial in all the unique combinations of the terms in **vars** using **collect**. However, **collect** works best on simple sums (i.e., objects of Maple type `` `+` ``) as opposed to series (i.e., objects of Maple type **series**), so we first convert **Error** before collecting:

```
> whattype(Error);
  Error := convert(Error,`+`);
  whattype(Error);
  Error := collect(Error,vars,'distributed');
```

series

$$Error := \left(f(x,y(x)) - b_1 f(x,y(x)) - b_2 f(x,y(x))\right) h + \left(\frac{1}{2} D_1(f)(x,y(x)) + \frac{1}{2} D_2(f)(x,y(x)) f(x,y(x))\right.$$
$$\left. - b_2 \left(D_1(f)(x,y(x)) c_2 + D_2(f)(x,y(x)) \alpha_{2,1} f(x,y(x))\right)\right) h^2 + O(h^3)$$

`+`

$$Error := \left(\frac{1}{2} - b_2 \alpha_{2,1}\right) D_2(f)(x,y(x)) h^2 f(x,y(x)) + \left(\frac{1}{2} - b_2 c_2\right) D_1(f)(x,y(x)) h^2 + (1 - b_1 - b_2) f(x, \quad (2.15)$$
$$y(x)) h + O(h^3)$$

The **'distributed'** option ensures that the coefficients of the resulting polynomial will only involve elements of the **unknowns** set. We can put each of the coefficients in the polynomial into a set using **coeffs**:

```
> sys := {coeffs(Error,vars)};
```

$$sys := \left\{1, \frac{1}{2} - b_2 \alpha_{2,1}, \frac{1}{2} - b_2 c_2, 1 - b_1 - b_2\right\} \quad (2.16)$$

Actually, this set includes the coefficient of $O(h^3)$, which is just 1. We're not really interested in this, so we remove it from the list. After this, we set each coefficient equal to zero to yield a system of equations for $\{\alpha_{n,m}, b_n, c_n\}$.

```
> sys := sys minus {1};
  sys := map(x->x=0,sys);
```

$$sys := \left\{\frac{1}{2} - b_2 \alpha_{2,1}, \frac{1}{2} - b_2 c_2, 1 - b_1 - b_2\right\}$$

$$sys := \left\{\frac{1}{2} - b_2 \alpha_{2,1} = 0, \frac{1}{2} - b_2 c_2 = 0, 1 - b_1 - b_2 = 0\right\} \quad (2.17)$$

We can try to attempt to solve this system:

```
> sol := solve(sys);
```

$$sol := \left\{\alpha_{2,1} = \frac{1}{2 b_2}, b_1 = 1 - b_2, b_2 = b_2, c_2 = \frac{1}{2 b_2}\right\} \quad (2.18)$$

We actually fail to get a definate solution, we rather get a one-parameter family of solutions. This is because the system of equations is actually underdetermined; there are many possible solutions. All of these solutions will generate a valid, explicit 2-stage Runge-Kutta stencil, which can be seen if we put **sol** into **Error**:

```
> subs(sol,Error);
```

$$O(h^3) \quad (2.19)$$

Stated another way: there is no unique 2-stage explicit Runge-Kutta method. However, to use the method we do need actual values for $\{\alpha_{n,m}, b_n, c_n\}$, so we need to make a choice. The customary additional assumption is $b_2 = \frac{1}{2}$, which yields a definitive answer for the other constants:

```
> assumptions := {b[2]=1/2};
  solution := solve(subs(assumptions,sys));
  answer := assumptions union solution;
```

$$assumptions := \left\{b_2 = \frac{1}{2}\right\}$$

$$solution := \left\{\alpha_{2,1} = 1, b_1 = \frac{1}{2}, c_2 = 1\right\}$$

$$answer := \left\{\alpha_{2,1} = 1, b_1 = \frac{1}{2}, b_2 = \frac{1}{2}, c_2 = 1\right\} \quad \textbf{(2.20}$$

This is known as the Heun or modified Euler stencil. Its Butcher tableau is

```
> subs(answer,Butcher);
```

$$\begin{bmatrix} & & \\ 1 & 1 & \\ & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \quad \textbf{(2.21}$$

Finally, the actual equations one uses in a numerical method are:

```
> for n from 1 to M do:
     subs(answer,k_def[n]):
  od;
  subs(answer,evolve);
```

$$k_1 = f\left(x_i, y_i\right)$$

$$k_2 = f\left(x_i + h, y_i + h\,k_1\right)$$

$$y_{i+1} = y_i + h\left(\frac{1}{2}k_1 + \frac{1}{2}k_2\right) \quad \textbf{(2.22}$$

And the local error is

```
> subs(answer,Error);
```

$$O\left(h^3\right) \quad \textbf{(2.23}$$

Notice that the Huen method gives the same one-step accuracy as the trapezoidal method, but it is fully explicit.

The classic fourth order Runge-Kutta scheme

```
> restart;
  Digits := 14;
```

$$Digits := 14 \qquad \textbf{(3.1}$$

We now turn our attention to an explicit 4-stage Runge-Kutta method. Note that the code from the previous section works for any value of M, so we could just re-run it to obtain the equations satisfied by $\{\alpha_{n,m}, b_n, c_n\}$ when $M = 4$. But we can take this opportunity to re-write the code a little more compactly:

```
> M := 4;

  for n from 1 to M do:
       for m from n to M do:
          alpha[n,m] := 0:
       od;
  od:

  c[1] := 0:

  for n from 1 to M do:
     k_def[n] := k[n] = f(x[i]+c[n]*h,y[i]+h*add(alpha[n,m]*k[m],m=1..M)):
  od:

  evolve := y[i+1] = y[i] + h*add(b[n]*k[n],n=1..M):
  unknowns := {seq(seq(alpha[n,m],m=1..n-1),n=2..M),seq(b[n],n=1..M),seq(c[n],n=2..M)}:

  derivative[1] := D(y)(x) = f(x,y(x)):
  for j from 2 to M do:
     derivative[j] := subs(derivative[1],convert(diff(derivative[j-1],x),D)):
  od:

  Subs := {y[i] = y(x),y[i+1] = y(x+h), x[i] = x}:
  Error := subs(convert(derivative,list),convert(series(subs(convert(k_def,list)$M,Subs,
  (lhs-rhs)(evolve)),h,M+1),`+`)):
  vars := indets(Error) minus unknowns minus {x,y(x)}:
  sys := {coeffs(collect(Error,vars,'distributed'),vars)} minus {1}:
  sys := map(x->simplify(x=0),sys);
```

$$M := 4$$

$$sys := \left\{ \frac{1}{24} - b_4\,\alpha_{4,3}\,\alpha_{3,2}\,\alpha_{2,1} = 0,\ \frac{1}{24} - b_4\,\alpha_{4,3}\,\alpha_{3,2}\,c_2 = 0,\ \frac{1}{24} - \frac{1}{6}\,b_4\,c_4^3 - \frac{1}{6}\,b_2\,c_2^3 - \frac{1}{6}\,b_3\,c_3^3 = 0,\ -b_2\,c_2 - b_3\,c_3 \right. \quad \textbf{(3.2}$$

$$- b_4\,c_4 + \frac{1}{2} = 0,\ -\frac{1}{2}\,b_2\,c_2^2 + \frac{1}{6} - \frac{1}{2}\,b_4\,c_4^2 - \frac{1}{2}\,b_3\,c_3^2 = 0,\ -\frac{1}{2}\,b_3\,\alpha_{3,2}\,c_2^2 + \frac{1}{24} - \frac{1}{2}\,b_4\,\alpha_{4,2}\,c_2^2 - \frac{1}{2}\,b_4\,\alpha_{4,3}\,c_3^2$$

$$= 0,\ -b_4\,\alpha_{4,3}\,c_3 - b_4\,\alpha_{4,2}\,c_2 - b_3\,\alpha_{3,2}\,c_2 + \frac{1}{6} = 0,\ -b_3\,c_3\,\alpha_{3,2}\,c_2 + \frac{1}{8} - b_4\,\alpha_{4,2}\,c_2\,c_4 - b_4\,\alpha_{4,3}\,c_3\,c_4 = 0,\ 1 - b_1$$

$$- b_2 - b_3 - b_4 = 0,\ -b_4\,\alpha_{4,3}\,\alpha_{3,2} - b_4\,\alpha_{4,2}\,\alpha_{2,1} - b_4\,\alpha_{4,3}\,\alpha_{3,1} - b_3\,\alpha_{3,2}\,\alpha_{2,1} + \frac{1}{6} = 0,\ -b_2\,\alpha_{2,1} - b_3\,\alpha_{3,1}$$

$$- b_3\,\alpha_{3,2} - b_4\,\alpha_{4,1} - b_4\,\alpha_{4,2} - b_4\,\alpha_{4,3} + \frac{1}{2} = 0,\ -\frac{1}{2}\,b_2\,c_2^2\,\alpha_{2,1} + \frac{1}{8} - \frac{1}{2}\,b_3\,c_3^2\,\alpha_{3,2} - \frac{1}{2}\,b_3\,c_3^2\,\alpha_{3,1}$$

$$- \frac{1}{2}\,b_4\,\alpha_{4,2}\,c_4^2 - \frac{1}{2}\,b_4\,c_4^2\,\alpha_{4,3} - \frac{1}{2}\,b_4\,\alpha_{4,1}\,c_4^2 = 0,\ -b_3\,c_3\,\alpha_{3,1} - b_3\,c_3\,\alpha_{3,2} - b_2\,c_2\,\alpha_{2,1} + \frac{1}{3} - b_4\,\alpha_{4,3}\,c_4$$

$$- b_4\,\alpha_{4,1}\,c_4 - b_4\,\alpha_{4,2}\,c_4 = 0,\ \frac{5}{24} - b_3\,c_3\,\alpha_{3,2}\,\alpha_{2,1} - b_3\,\alpha_{3,2}\,c_2\,\alpha_{2,1} - b_4\,\alpha_{4,3}\,\alpha_{3,1}\,c_4 - b_4\,\alpha_{4,2}\,c_2\,\alpha_{2,1}$$

$$- b_4\,\alpha_{4,3}\,\alpha_{3,2}\,c_4 - b_4\,\alpha_{4,2}\,\alpha_{2,1}\,c_4 - b_4\,\alpha_{4,3}\,c_3\,\alpha_{3,1} - b_4\,\alpha_{4,3}\,c_3\,\alpha_{3,2} = 0,\ -b_4\,\alpha_{4,3}^2\,c_3 - b_4\,\alpha_{4,1}\,\alpha_{4,3}\,c_3$$

$$- b_4\,\alpha_{4,1}\,\alpha_{4,2}\,c_2 - b_4\,\alpha_{4,2}^2\,c_2 - b_4\,\alpha_{4,2}\,\alpha_{4,3}\,c_3 - b_4\,\alpha_{4,3}\,\alpha_{4,2}\,c_2 - b_3\,\alpha_{3,1}\,\alpha_{3,2}\,c_2 - b_3\,\alpha_{3,2}^2\,c_2 + \frac{1}{8} = 0,\ -\frac{1}{2}\,b_4$$

$$\alpha_{4,2}^2 - \frac{1}{2}\,b_4\,\alpha_{4,1}^2 - \frac{1}{2}\,b_4\,\alpha_{4,3}^2 - b_4\,\alpha_{4,1}\,\alpha_{4,2} - b_4\,\alpha_{4,1}\,\alpha_{4,3} - b_4\,\alpha_{4,2}\,\alpha_{4,3} + \frac{1}{6} - b_3\,\alpha_{3,1}\,\alpha_{3,2} - \frac{1}{2}\,b_3\,\alpha_{3,1}^2$$

$$- \frac{1}{2}\,b_3\,\alpha_{3,2}^2 - \frac{1}{2}\,b_2\,\alpha_{2,1}^2 = 0,\ -\frac{1}{2}\,b_3\,c_3\,\alpha_{3,1}^2 - b_3\,\alpha_{3,1}\,\alpha_{3,2}\,c_3 - \frac{1}{2}\,b_3\,c_3\,\alpha_{3,2}^2 - b_4\,\alpha_{4,2}\,c_4\,\alpha_{4,3} - \frac{1}{2}\,b_4\,\alpha_{4,1}^2\,c_4$$

$$- \frac{1}{2}\,b_4\,\alpha_{4,2}^2\,c_4 - b_4\,\alpha_{4,1}\,c_4\,\alpha_{4,2} - b_4\,\alpha_{4,1}\,c_4\,\alpha_{4,3} - \frac{1}{2}\,b_4\,\alpha_{4,3}^2\,c_4 + \frac{1}{8} - \frac{1}{2}\,b_2\,c_2\,\alpha_{2,1}^2 = 0,\ -\frac{1}{6}\,b_2\,\alpha_{2,1}^3$$

$$- \frac{1}{2}\,b_3\,\alpha_{3,1}\,\alpha_{3,2}^2 - \frac{1}{2}\,b_3\,\alpha_{3,1}^2\,\alpha_{3,2} - \frac{1}{6}\,b_3\,\alpha_{3,1}^3 - \frac{1}{6}\,b_3\,\alpha_{3,2}^3 + \frac{1}{24} - \frac{1}{2}\,b_4\,\alpha_{4,2}^2\,\alpha_{4,1} - \frac{1}{2}\,b_4\,\alpha_{4,2}^2\,\alpha_{4,3} - \frac{1}{2}\,b_4$$

$$\alpha_{4,1}^2\,\alpha_{4,3} - \frac{1}{6}\,b_4\,\alpha_{4,2}^3 - b_4\,\alpha_{4,1}\,\alpha_{4,2}\,\alpha_{4,3} - \frac{1}{2}\,b_4\,\alpha_{4,3}^2\,\alpha_{4,1} - \frac{1}{2}\,b_4\,\alpha_{4,3}^2\,\alpha_{4,2} - \frac{1}{2}\,b_4\,\alpha_{4,1}^2\,\alpha_{4,2} - \frac{1}{6}\,b_4\,\alpha_{4,1}^3$$

$$-\frac{1}{6}\,b_4\,\alpha_{4,3}^3 = 0,\ \frac{1}{6} - b_3\,\alpha_{3,1}\,\alpha_{3,2}\,\alpha_{2,1} - \frac{1}{2}\,b_3\,\alpha_{3,2}\,\alpha_{2,1}^2 - b_3\,\alpha_{3,2}^2\,\alpha_{2,1} - b_4\,\alpha_{4,3}^2\,\alpha_{3,1} - b_4\,\alpha_{4,2}^2\,\alpha_{2,1}$$

$$- b_4\,\alpha_{4,3}\,\alpha_{3,1}\,\alpha_{3,2} - b_4\,\alpha_{4,1}\,\alpha_{4,3}\,\alpha_{3,1} - b_4\,\alpha_{4,2}\,\alpha_{4,3}\,\alpha_{3,2} - b_4\,\alpha_{4,1}\,\alpha_{4,2}\,\alpha_{2,1} - \frac{1}{2}\,b_4\,\alpha_{4,3}\,\alpha_{3,1}^2 - b_4\,\alpha_{4,3}^2\,\alpha_{3,2}$$

$$- \frac{1}{2}\,b_4\,\alpha_{4,3}\,\alpha_{3,2}^2 - b_4\,\alpha_{4,1}\,\alpha_{4,3}\,\alpha_{3,2} - \frac{1}{2}\,b_4\,\alpha_{4,2}\,\alpha_{2,1}^2 - b_4\,\alpha_{4,3}\,\alpha_{4,2}\,\alpha_{2,1} - b_4\,\alpha_{4,2}\,\alpha_{4,3}\,\alpha_{3,1} = 0\Big\}$$

This is the set of equations that must be satisfied by $\{\alpha_{n,m}, b_n, c_n\}$ to have a valid 4-stage stencil. We can now ask Maple to solve this system of equations. I have supressed the output as it is very long. (Warning: the following step takes about 30 seconds on my machine. I you execute it on your computer, the number printed out is how much CPU time you used. See **?time()** for more details.)

```
> st := time():
  rk4_solutions := {solve(sys)}:
  time()-st;
```

$$28.386 \qquad \textbf{(3.3}$$

The number of classes of solutions returned by **solve** can be obtained using the **nops** command:

```
> NN := nops(rk4_solutions);
```

$$NN := 7 \qquad \textbf{(3.4}$$

And here is an example of class of solutions:

```
> rk4_solutions[1];
```

$$\Big\{\alpha_{2,1} = \frac{1}{2}, \alpha_{3,1} = -\frac{1}{2\,\alpha_{4,3}}, \alpha_{3,2} = \frac{1}{2\,\alpha_{4,3}}, \alpha_{4,1} = -\frac{1}{2} - \alpha_{4,3}, \alpha_{4,2} = \frac{3}{2}, \alpha_{4,3} = \alpha_{4,3}, b_1 = \frac{1}{6} - \frac{1}{6}\,\alpha_{4,3}, b_2 = \frac{2}{3}, b_3 \qquad \textbf{(3.5}$$

$$= \frac{1}{6}\,\alpha_{4,3}, b_4 = \frac{1}{6}, c_2 = \frac{1}{2}, c_3 = 0, c_4 = 1\Big\}$$

This is a parameteric solution; i.e., just as in the $M = 2$ case, there are an infinite number of solutions for $\{\alpha_{n,m}, b_n, c_n\}$. The "classic fourth order Runge-Kutta method" involves the following assumptions:

```
> assumptions := {alpha[4, 1] = 0, alpha[3, 1] = 0, alpha[4, 2] = 0, b[4] = b[1], b[3] = b
  [2]};
```

$$assumptions := \{\alpha_{3,1} = 0, \alpha_{4,1} = 0, \alpha_{4,2} = 0, b_3 = b_2, b_4 = b_1\} \qquad \textbf{(3.6}$$

This leads to the following solution for the coefficients:

```
> sol := solve(subs(assumptions,sys));
  answer := sol union subs(sol,assumptions);
```

$$sol := \left\{\alpha_{2,1} = \frac{1}{2}, \alpha_{3,2} = \frac{1}{2}, \alpha_{4,3} = 1, b_1 = \frac{1}{6}, b_2 = \frac{1}{3}, c_2 = \frac{1}{2}, c_3 = \frac{1}{2}, c_4 = 1\right\}$$

$$answer := \left\{\alpha_{2,1} = \frac{1}{2}, \alpha_{3,1} = 0, \alpha_{3,2} = \frac{1}{2}, \alpha_{4,1} = 0, \alpha_{4,2} = 0, \alpha_{4,3} = 1, b_1 = \frac{1}{6}, b_2 = \frac{1}{3}, b_3 = \frac{1}{3}, b_4 = \frac{1}{6}, c_2 = \frac{1}{2}, c_3 = \frac{1}{2}, c_4 = 1\right\} \quad (3.7$$

The Butcher tableau for this stencil is

```
> A := Matrix([seq([seq(alpha[n,m],m=1..M)],n=1..M)]):
  B := Vector[row]([seq(b[n],n=1..M)]):
  C := Vector[column]([seq(c[n],n=1..M)]):
  Butcher = subs(answer,map(x->if (x=0) then `` else x end if,Matrix([[C,A],[``,B]])));
```

$$Butcher = \begin{bmatrix} & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 0 & 0 & 1 & \\ & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{bmatrix} \quad (3.8$$

Here are the basic equations

```
> rk4 := subs(answer,[seq(k_def[n],n=1..M),evolve]);
```

$$rk4 := \left[k_1 = f(x_i, y_i), k_2 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}h\,k_1\right), k_3 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}h\,k_2\right), k_4 = f\left(x_i + h, y_i + h\,k_3\right), y_{i+1} = y_i + h\left(\frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4\right)\right] \quad (3.9$$

We can confirm that the one-step error is $O(h^{M+1})$ as expected:

```
> simplify(subs(answer,Error));
```

$$O(h^5) \quad (3.10$$

We conclude by writing a procedure that makes use of the classic Runge-Kutta method to solve $y'(x) = f(x, y(x))$. We first transform the **rk4** stencil into a set of mappings:

```
> for n from 1 to M+1 do:
      K[n] := unapply(rhs(rk4[n]),h,x[i],y[i],seq(k[m],m=1..n-1));
  od;
```

$$K_1 := (h, y2, y3) \rightarrow f(y2, y3)$$

$$K_2 := (h, y2, y3, k\_1) \rightarrow f\left(y2 + \frac{1}{2} h, y3 + \frac{1}{2} h\, k\_1\right)$$

$$K_3 := (h, y2, y3, k\_1, k\_2) \rightarrow f\left(y2 + \frac{1}{2} h, y3 + \frac{1}{2} h\, k\_2\right)$$

$$K_4 := (h, y2, y3, k\_1, k\_2, k\_3) \rightarrow f(h + y2, y3 + h\, k\_3)$$

$$K_5 := (h, y2, y3, k\_1, k\_2, k\_3, k\_4) \rightarrow y3 + h\left(\frac{1}{6} k\_1 + \frac{1}{3} k\_2 + \frac{1}{3} k\_3 + \frac{1}{6} k\_4\right) \qquad \textbf{(3.11}$$

Here is a procedure that uses these to calculate a numeric solution in the interval $x \in [0, 1]$ using N steps and with initial data $y(0) = y0$:

```
> RK4 := proc(y0,N)
       local h, x, y, i, k:

       h := evalf(1/N):
       x[0] := 0:
       y[0] := y0:
       for i from 1 to N do:
          x[i] := x[i-1] + h:
          k[1] := K[1](h,x[i-1],y[i-1]):
          k[2] := K[2](h,x[i-1],y[i-1],k[1]):
          k[3] := K[3](h,x[i-1],y[i-1],k[1],k[2]):
          k[4] := K[4](h,x[i-1],y[i-1],k[1],k[2],k[3]):
          y[i] := K[5](h,x[i-1],y[i-1],k[1],k[2],k[3],k[4]):
       od:
       [seq([x[i],y[i]],i=0..N)]:

  end proc:
```

We'll compare the fourth order Runge-Kutta results with those obtained by the forward Euler stencil as calculated by **EulerSol**:

```
> Euler := proc(y0,N)
       local h, x, y, i:

       h := evalf(1/N):
       x[0] := 0:
       y[0] := y0:
```

```
      for i from 1 to N do:
         x[i] := x[i-1] + h:
         y[i] := y[i-1] + h*f(x[i-1],y[i-1]):
      od:
      [seq([x[i],y[i]],i=0..N)]:

  end proc:
```

Here is a plot of the output generated by the two stencils compare to the analytic solution of the equation for a particular choice of $f(x,y)$.

```
> f := (x,y) -> -y^2+2*x;
  y0 := 2;
  N := 5;
  ode := diff(y(x),x)=f(x,y(x));
  analytic_sol := rhs(dsolve({ode,y(0)=y0}));

  plot([Euler(y0,N),RK4(y0,N),analytic_sol],x=0..1,style=[point$2,line],legend=[`Forward
  Euler`,`Runge Kutta 4th order`,`analytic solution`],axes=boxed);
```

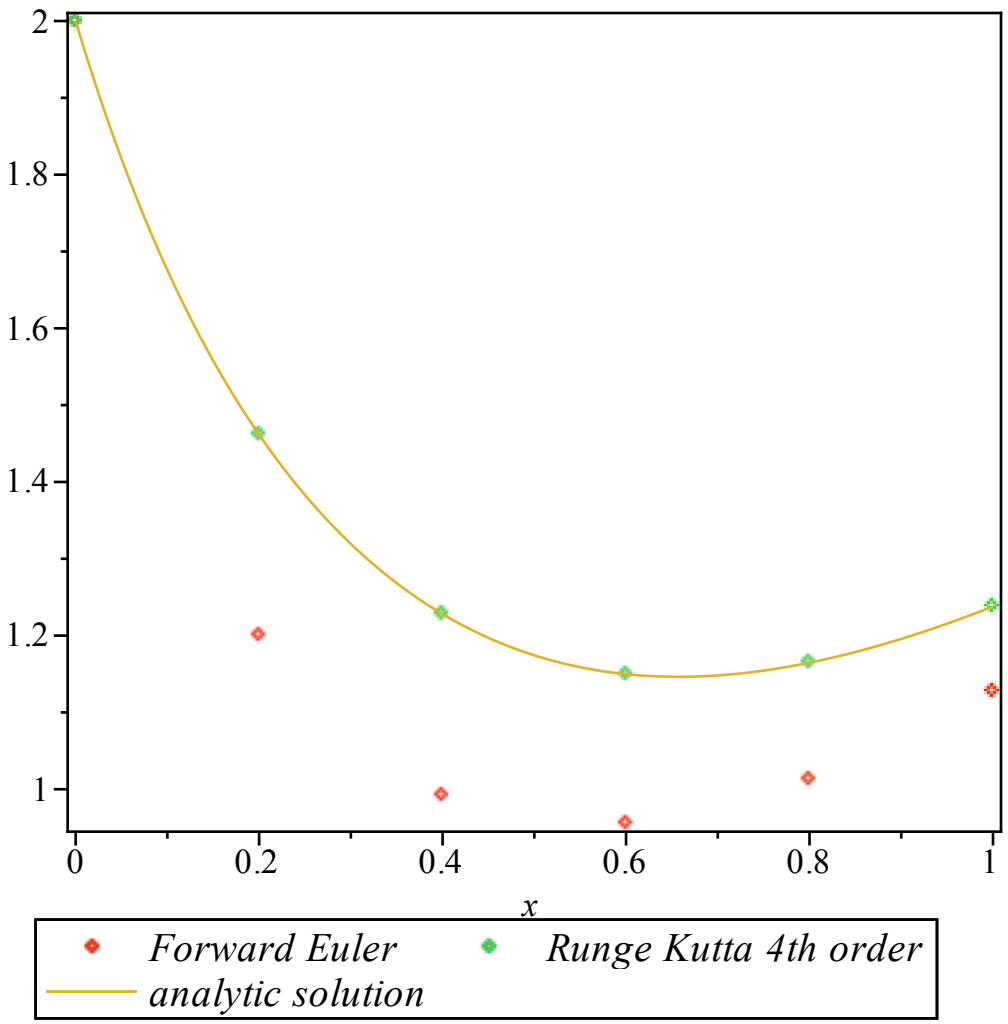
$$f := (x,y) \to -y^2 + 2x$$

$$y0 := 2$$

$$N := 5$$

$$ode := \frac{d}{dx} y(x) = -y(x)^2 + 2x$$

$$analytic\_sol := \frac{2^{1/3}\left(\frac{\left(-4\pi 3^{5/6} + 3\,2^{1/3}\Gamma\left(\frac{2}{3}\right)^2 3^{2/3}\right)\mathrm{AiryAi}\left(1, 2^{1/3}x\right)}{4\pi 3^{1/3} + 3\,2^{1/3}\Gamma\left(\frac{2}{3}\right)^2 3^{1/6}} + \mathrm{AiryBi}\left(1, 2^{1/3}x\right)\right)}{\frac{\left(-4\pi 3^{5/6} + 3\,2^{1/3}\Gamma\left(\frac{2}{3}\right)^2 3^{2/3}\right)\mathrm{AiryAi}\left(2^{1/3}x\right)}{4\pi 3^{1/3} + 3\,2^{1/3}\Gamma\left(\frac{2}{3}\right)^2 3^{1/6}} + \mathrm{AiryBi}\left(2^{1/3}x\right)}$$



We see for even a moderate number of steps, the agreement between the Runge-Kutta method and the analytic solution is remarkable. We can quantify just how much better the Runge-Kutta stencil does by defining a measure of the global error ε as the magnitude of the discrepancy between the numerical and actual values of $y(1)$ for each stencil:

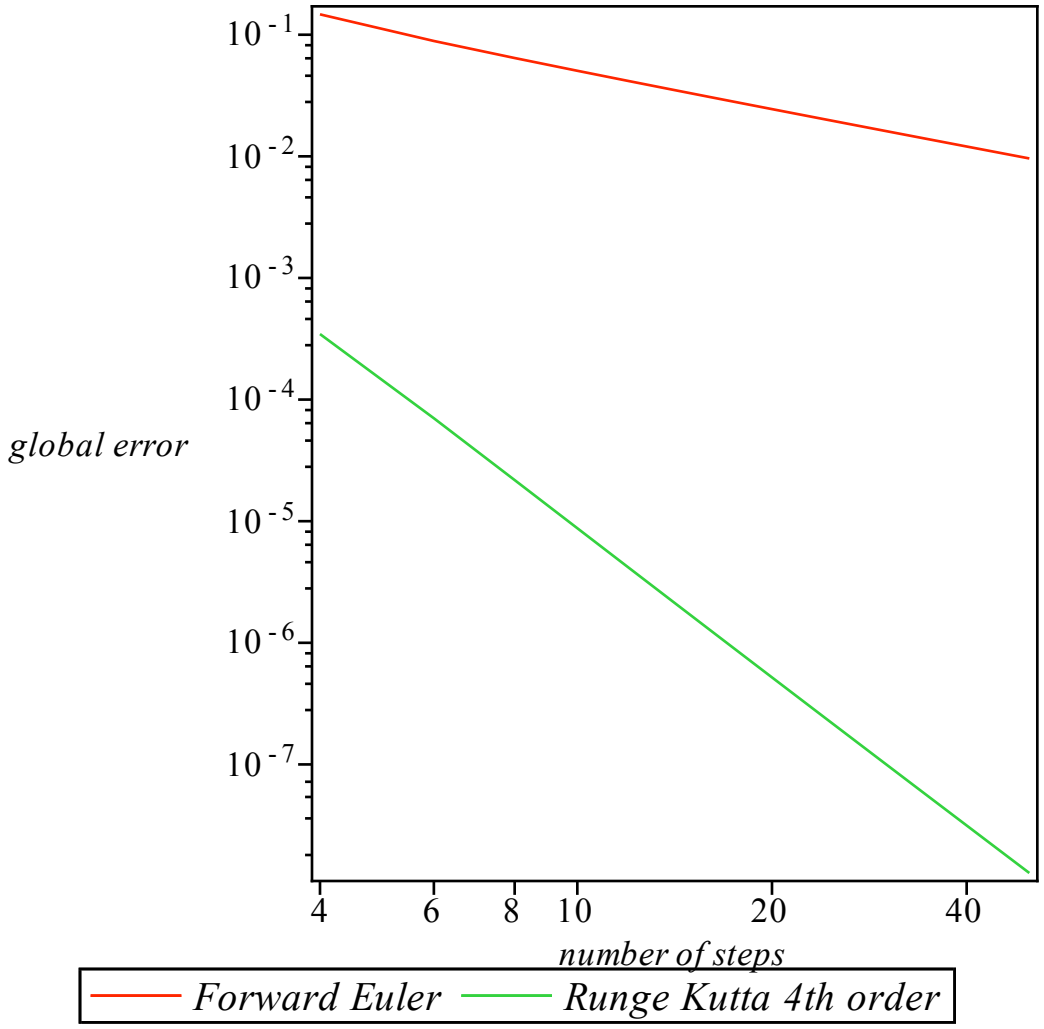
```
> epsilon := (y0,N,Method) -> evalf(abs(Method(y0,N)[N+1][2]-eval(analytic_sol,x=1)));
```

$$\varepsilon := (y0, N, Method) \rightarrow evalf\left(\left|Method(y0, N)_{1+N_2} - \left(analytic\_sol\Big|_{x=1}\right)\right|\right) \quad \textbf{(3.12}$$

Here is a log-log plot of ε as a function of the number of steps *N*:

```
> data[RK4] := [seq([N,epsilon(y0,N,RK4)],N=4..50,2)]:
  data[Euler] := [seq([N,epsilon(y0,N,Euler)],N=4..50,2)]:

  plots[loglogplot]([data[Euler],data[RK4]],legend=[`Forward Euler`,`Runge Kutta 4th
  order`],axes=boxed,labels=[`number of steps`,`global error`]);
```



Clearly, the error for the Runge-Kutta method is several orders of magnitude lower. Futhermore, the global error curves both look linear on

the log-log plot, which suggests that there is a power law dependence of ε on N: $\varepsilon \sim \varepsilon_0 N^{-a} \propto h^a$. We can determine the power a by fitting a power law to the data using `Statistics[PowerFit]`:

```
> `numerically determined global error (Forward Euler) = ` || O(h^(-Statistics[PowerFit]
  (convert(data[Euler],Matrix))[2]));
  `numerically determined global error (Runge-Kutta 4th order) = ` || O(h^(-Statistics
  [PowerFit](convert(data[RK4],Matrix))[2]));
```

$$\textit{numerically determined global error (Forward Euler)} = O\left(h^{1.05698315495806}\right)$$

$$\textit{numerically determined global error (Runge-Kutta 4th order)} = O\left(h^{4.05310492580892}\right) \quad \textbf{(3.13}$$

This matches our expectation that if the one-step error in a stencil is $O\left(h^{p+1}\right)$, the global error ought to be $O\left(h^p\right)$ for $N \gg 1$ [see **(3.10)** above].

NOTE: the above syntax for `Statistics[PowerFit]` only works in Maple 15. For previous versions, you need to do this:

```
> with(LinearAlgebra):
  M1,M2 := convert(data[Euler],Matrix),convert(data[RK4],Matrix):
  X1,X2 := Column(M1,1),Column(M2,1):
  Y1,Y2 := Column(M1,2),Column(M2,2):
  `numerically determined global error (Forward Euler) = ` || O(h^(-Statistics[PowerFit](X1,
  Y1)[2]));
  `numerically determined global error (Runge-Kutta 4th order) = ` || O(h^(-Statistics
  [PowerFit](X2,Y2)[2]));
```

$$\textit{numerically determined global error (Forward Euler)} = O\left(h^{1.05698315495806}\right)$$

$$\textit{numerically determined global error (Runge-Kutta 4th order)} = O\left(h^{4.05310492580892}\right) \quad \textbf{(3.14}$$