

## MATH 231A

### Runge-Kutta Notes: for a 2-equation, 1st-order system

For our (general) problem from class

$$\begin{aligned}dx/dt &= f(t, x, y), & x(t_0) &= x_0 \\dy/dt &= g(t, x, y), & y(t_0) &= y_0\end{aligned}$$

we get our approximate solution  $(x_n, y_n)$  at time  $t_n$ ,  $n = 1, 2, \dots$  via the iteration of

$$\begin{aligned}x_{n+1} &= x_n + \frac{h}{6}(k_{n1} + 2k_{n2} + 2k_{n3} + k_{n4}) \\y_{n+1} &= y_n + \frac{h}{6}(\ell_{n1} + 2\ell_{n2} + 2\ell_{n3} + \ell_{n4}),\end{aligned}$$

where the formulas for each of the  $k$ 's and  $\ell$ 's are

$$\begin{aligned}k_{n1} &= f(t_n, x_n, y_n) \\ \ell_{n1} &= g(t_n, x_n, y_n) \\ k_{n2} &= f(t_n + h/2, x_n + \frac{1}{2}hk_{n1}, y_n + \frac{1}{2}h\ell_{n1}) \\ \ell_{n2} &= g(t_n + h/2, x_n + \frac{1}{2}hk_{n1}, y_n + \frac{1}{2}h\ell_{n1}) \\ k_{n3} &= f(t_n + h/2, x_n + \frac{1}{2}hk_{n2}, y_n + \frac{1}{2}h\ell_{n2}) \\ \ell_{n3} &= g(t_n + h/2, x_n + \frac{1}{2}hk_{n2}, y_n + \frac{1}{2}h\ell_{n2}) \\ k_{n4} &= f(t_n + h, x_n + hk_{n3}, y_n + h\ell_{n3}) \\ \ell_{n4} &= g(t_n + h, x_n + hk_{n3}, y_n + h\ell_{n3}).\end{aligned}$$

#### Note about reading Section 8.6

When you read through Section 8.6, the formulas that appear there look different only because, in the text,  $\mathbf{x}_n$  and  $\mathbf{k}_{nj}$  are treated as vectors. Formulas (6) and (7) on p. 456 become the same as above when we note that, in the 2-dependent-variable case, the way to translate the authors' notation into ours is

$$\mathbf{x}_n = (x_n, y_n) \quad \text{and} \quad \mathbf{k}_{nj} = (k_{nj}, \ell_{nj}).$$

In the *Mathematica* notebook that you will download (in which there is a Runge-Kutta algorithm for the two-body problem), you will see that I have written the algorithm in two different ways, the first time in scalar form (i.e., like the formulas above, only applied to 4 dependent variables instead of 2), and once in vector form. Two things I would like to point out:

- While, at each step in the algorithm, we need to store a new 4-tuple  $(x, u, y, v)$  describing the current *state* of the satellite (or, at least  $(x, y)$ , which are enough to give the satellite's position), the computation of the next  $(x_{n+1}, u_{n+1}, y_{n+1}, v_{n+1})$  only requires knowledge of the current  $(x_n, u_n, y_n, v_n)$ . A statement (taken from the notebook you download) like

$$x = x + h * (j1 + 2 * j2 + 2 * j3 + j4)/6;$$

uses the old  $x$  value on the right side of the equation, and the result of this computation is then stored as  $x$  (wiping out the old  $x$ -value). The same may be said about the numbers  $j_1, \dots, j_4, k_1, \dots, k_4, \dots, m_1, \dots, m_4$ .

- In *Mathematica*, points and vectors are considered to be roughly the same thing; both are stored as lists, surrounded by curly brackets. The function definition (found in the vector/2nd version of the algorithm in the notebook)

$$f[t_, x_] := \{x[[2]], -x[[1]]/(x[[1]]^2 + x[[3]]^2)^{3/2}, x[[4]], -x[[3]]/(x[[1]]^2 + x[[3]]^2)^{3/2}\}$$

takes a scalar  $t$  (which it doesn't really use) and a 4-dimensional vector  $\mathbf{x}$ . This functions output is also a 4-dimensional vector (note the curly brackets and the 4 separate components lying between them).