Queens College, CUNY, Department of Computer Science
Numerical Methods
CSCI 361 / 761
Spring 2018
Instructor: Dr. Sateesh Mane

April 29, 2018

20 Lecture 20

Numerical solution of systems of ordinary differential equations

- In this lecture we continue the study of initial value problems.
- In this lecture we shall study the technique of **adaptive stepsize control.**
- Although we have said that the integration stepsize h_i can depend on the step i, up to now all of our examples have employed a uniform stepsize.
- In this lecture we shall study techniques where the stepsize varies with each step.
- The technique of adaptive stepsize control can be applied to any numerical integration algorithm, but we shall treat only Runge–Kutta methods.

20.1 Basic notation

- We repeat the basic definitions from the previous lecture.
- Let the system of coupled differential equations be

$$\frac{d\boldsymbol{y}}{dx} = \boldsymbol{f}(x, \boldsymbol{y})\,. \tag{20.1.1}$$

- There are m unknown variables y_j, $j = 1, \ldots, m$.
- The starting point is $x = x_0$, and the initial value $\boldsymbol{y}_0$ is given.
- The above is called the **Cauchy problem** or **initial value problem.**
- Our interest is to integrate eq. (20.1.1) numerically, using steps h_i, so $x_{i+1} = x_i + h_i$.
- The steps h_i need not be of equal size.
- We define $\boldsymbol{y}_i = \boldsymbol{y}(x_i)$.
- We employ the notation $\boldsymbol{y}_i^{\text{ex}} = \boldsymbol{y}^{\text{ex}}(x_i)$ to denote the exact solution.
- We employ the notation $\boldsymbol{y}_i^{\text{num}} = \boldsymbol{y}^{\text{num}}(x_i)$ to denote the numerical solution.

20.2 Motivating example 1

- Consider the following differential equation, where $\sigma > 0$ is a positive constant:

$$\frac{dy}{dx} = -\frac{xy}{\sigma} \,. \tag{20.2.1}$$

- The initial condition is $y = 1$ at $x = 0$.
- The solution is a Gaussian

$$y(x) = e^{-x^2/(2\sigma^2)} \,. \tag{20.2.2}$$

- An example Gaussian is plotted in Fig. 1.
- Note that the value of $y(x)$ is peaked around $x = 0$ and is almost constant (in fact almost zero) for $|x| \gg \sigma$.
- If the value of σ is small, the solution $y(x)$ varies rapidly in a small interval around $x = 0$.
- Suppose we integrated eq. (20.2.1) with a uniform stepsize h.
  1. To obtain a good numerical approximation, we would require $h \ll \sigma$.
  2. However, a stepsize $h \ll \sigma$ is wasteful of computation time in the region $|x| \gg \sigma$.
  3. The solution is almost constant (in fact almost zero) when $|x| \gg \sigma$.
- Hence for this example it is preferable to employ a small stepsize if $|x| \lesssim \sigma$ and to employ a larger stepsize if $|x| \gg \sigma$.
- In general, if the solution of a system of coupled differential equation varies rapidly in some region, it is preferable to employ a small stepsize when traversing that region. In regions where the solution is varying only slowly (hence is approximately constant), we can employ a larger stepsize without losing numerical accuracy.

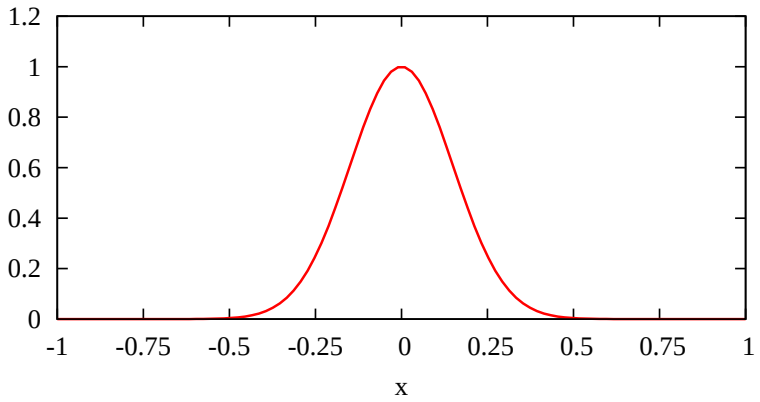

Figure 1: Graph of Gaussian function $e^{-x^2/(2\sigma^2)}$.

20.3 Motivating example 2

- Consider the motion of a planet around the Sun, or a satellite around a parent body.
- Suppose the motion is governed by an inverse–square force law, i.e. (k is a positive constant)

$$\frac{d^2\boldsymbol{r}}{dt} = -\frac{k}{|\boldsymbol{r}|^2}\,. \tag{20.3.1}$$

- This is a second–order differential equation (with vectors), but with the aid of suitable auxiliary variables it can be expressed in the form of eq. (20.1.1). The details are not important for the present discussion.
- A sketch of the orbit of the satellite is plotted in Fig. 2.
    1. Clearly, in regions where the orbit has a high curvature, a small stepsize is required, to compute the orbit accurately.
    2. In regions where the orbit has a low curvature, a larger stepsize can be used to compute the orbit, without losing numerical accuracy.
    3. Using a small stepsize for all the integration steps is wasteful of computating time.

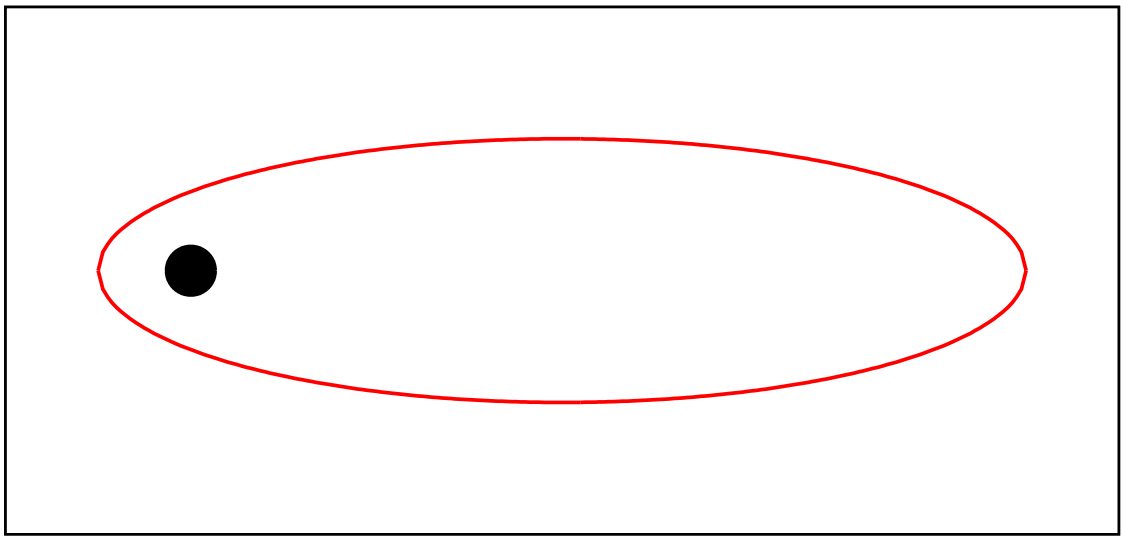

Figure 2: Graph of orbit of satellite around a parent object.

20.4 Estimate of local error

- For simplicity, suppose there is only one unknown and call it y.
- To begin, suppose that we employ a uniform stepsize h.
- Suppose we use an integration algorithm of order k, so the local truncation error is $O(h^{k+1})$.
- This means that if the exact solution is $y^{\text{ex}}(x)$, then

$$y^{\text{ex}}(x_i) = y_i + h^{k+1}\phi + \cdots . \tag{20.4.1}$$

- Here ϕ is some unknown but bounded function. It is constant over the integration step.
- For brevity, denote the local error by Δ, so

$$\Delta = h^{k+1}\phi . \tag{20.4.2}$$

- Suppose we have *another* integration algorithm (with the same stepsize), but of order $k+1$.
- Denote the numerical solution from the second algorithm by z_i. Then

$$y^{\text{ex}}(x_i) = z_i + h^{k+2}\tilde{\phi} + \cdots . \tag{20.4.3}$$

- Here $\tilde{\phi}$ is also some unknown but bounded function. Also define $\tilde{\Delta} = h^{k+2}\tilde{\phi}$.
- Note that we know the values of y_i and z_i. Then we can subtract eqs. (20.4.1) and (20.4.3) to obtain

$$z_i - y_i = \Delta + O(h^{k+2}) . \tag{20.4.4}$$

- By hypotheses, the terms of $O(h^{k+2})$ in eq. (20.4.4) are negligible. Hence we estimate the value of Δ via

$$\Delta = z_i - y_i . \tag{20.4.5}$$

- Note the following:
  1. We require *two* numerical integrators.
  2. This of course adds to the computational cost. This will be discussed below.
  3. Using two integrators, we can estimate the numerical error Δ in the integration step.
  4. Hence we not only have a numerical value for the solution, i.e. y_i but we also have an estimate how accurate it is, i.e. Δ.
  5. We also have the value z_i. This will be discussed below.

20.5 Adaptive stepsize control

- We continue the analysis in Sec. 20.4.
- Suppose we set a tolerance on the global absolute error.
- Suppose the tolerance is an error of not more that ε after N integration steps.
- Then for one step the tolerance on the numerical error is ε/N.
- For later use below define $\Delta_0 = \varepsilon/N$.
- Then since $\Delta \propto h^{k+1}$, we can define a parameter h_0 such that

$$\frac{h_0^{k+1}}{|\Delta_0|} = \frac{h^{k+1}}{|\Delta|} \,. \tag{20.5.1}$$

- Basically, h_0 is the tolerance on the stepsize we can take to attain a numerical error of Δ_0.
- Process eq. (20.5.1) to obtain

$$h_0 = h \left| \frac{\Delta_0}{\Delta} \right|^{1/(k+1)} \,. \tag{20.5.2}$$

- All the quantities on the right-hand side of eq. (20.5.2) are known.
- Now we analyze as follows:

  1. If $|\Delta| < |\Delta_0|$, then $h_0 > h$ and eq. (20.5.2) tells us that we can increase the stepsize from h to h_0 and still satisfy the error tolerance.
  2. In other words, it is safe to use a larger stepsize.
  3. In practice it might be safer to increase the value of h to slightly less than h_0.
  4. If $|\Delta| > |\Delta_0|$, then $h_0 < h$ and eq. (20.5.2) tells us that our stepsize is too large.
  5. We must recompute with a smaller stepsize and test again, to check that we satisfy the error tolerance.
  6. Remember that when we recompute with a different value of h, the value of Δ will also change and so we must test eq. (20.5.2) again.
  7. The procedure may have to be repeated several times, to determine an acceptable value for h.

- When there are multiple unknowns, the procedure is more complicated because Δ is a multi-component vector, one for each unknown.
- The tolerance criterion may be very different for the different unknowns.
- We have discussed only the **absolute tolerance** above.
- A **relative tolerance** may be more appropriate in some circumstances.

20.6 Embedded integrator

- We continue the analysis in Secs. 20.4 and 20.5.
- Recall from Sec. 20.4 that we require *two* numerical integrators.
- It is possible to formulate a fifth order Runge–Kutta integrator so that a fourth order Runge–Kutta integrator is "embedded" in it.
- The following scheme is called the **Cash–Karp method.**
- It is a Runge–Kutta method and has the acronym `RKCK`.
- We compute the following six function evaluations.

$$\boldsymbol{g}_1 = \boldsymbol{f}(x_i, \boldsymbol{z}_i)\,, \tag{20.6.1a}$$

$$\boldsymbol{g}_2 = \boldsymbol{f}\Big(x_i + \frac{h_i}{5}, \boldsymbol{z}_i + \frac{h_i}{5}\,\boldsymbol{g}_1\Big)\,, \tag{20.6.1b}$$

$$\boldsymbol{g}_3 = \boldsymbol{f}\Big(x_i + \frac{3h_i}{10}, \boldsymbol{z}_i + \frac{3h_i}{40}\,\boldsymbol{g}_1 + \frac{9h_i}{40}\,\boldsymbol{g}_2\Big)\,, \tag{20.6.1c}$$

$$\boldsymbol{g}_4 = \boldsymbol{f}\Big(x_i + \frac{3h_i}{5}, \boldsymbol{z}_i + \frac{3h_i}{10}\,\boldsymbol{g}_1 - \frac{9h_i}{10}\,\boldsymbol{g}_2 + \frac{6h_i}{5}\,\boldsymbol{g}_3\Big)\,, \tag{20.6.1d}$$

$$\boldsymbol{g}_5 = \boldsymbol{f}\Big(x_i + h_i, \boldsymbol{z}_i - \frac{11h_i}{54}\,\boldsymbol{g}_1 + \frac{5h_i}{2}\,\boldsymbol{g}_2 - \frac{70h_i}{27}\,\boldsymbol{g}_3 + \frac{35h_i}{27}\,\boldsymbol{g}_4\Big)\,, \tag{20.6.1e}$$

$$\boldsymbol{g}_6 = \boldsymbol{f}\Big(x_i + \frac{7h_i}{8}, \boldsymbol{z}_i + \frac{1631h_i}{55296}\,\boldsymbol{g}_1 + \frac{175h_i}{512}\,\boldsymbol{g}_2 + \frac{575h_i}{13824}\,\boldsymbol{g}_3 + \frac{44275h_i}{110592}\,\boldsymbol{g}_4 + \frac{253h_i}{4096}\,\boldsymbol{g}_5\Big)\,. \tag{20.6.1f}$$

- **We can construct TWO Runge–Kutta integrators using the above functions.**
- The fifth order Runge–Kutta integrator is given by

$$\boldsymbol{z}_{i+1} = \boldsymbol{z}_i + \frac{375h_i}{378}\,\boldsymbol{g}_1 + \frac{250h_i}{621}\,\boldsymbol{g}_3 + \frac{125h_i}{594}\,\boldsymbol{g}_4 + \frac{512h_i}{1771}\,\boldsymbol{g}_6\,. \tag{20.6.2}$$

- The fourth order Runge–Kutta integrator is given by

$$\boldsymbol{y}_{i+1} = \boldsymbol{z}_i + \frac{2825h_i}{27648}\,\boldsymbol{g}_1 + \frac{18575h_i}{48384}\,\boldsymbol{g}_3 + \frac{13525h_i}{55296}\,\boldsymbol{g}_4 + \frac{277h_i}{14336}\,\boldsymbol{g}_5 + \frac{h_i}{4}\,\boldsymbol{g}_6\,. \tag{20.6.3}$$

- Hence at a total cost of six function evaluations, we obtain both a fourth and a fifth order Runge–Kutta integrator.

20.7 Local extrapolation

- Note that the error estimate Δ is for the value of y_i.
- Nevertheless, the value of z_i is available, and it is presumably more accurate than y_i.
- Hence it makes sense to use the value of z_i to advance the numerical solution to the next step.
- This is called **local extrapolation.**