Numerical Simulation of Dynamic Systems: Hw3 - Solution

Prof. Dr. François E. Cellier
Department of Computer Science
ETH Zurich

March 19, 2013

[H3.4] RK Order Increase by Blending

Given two separate n^{th}-order accurate RK algorithms in at least $(n+1)$ stages:

$$\begin{aligned} f_1(q) &= 1 + q + \frac{q^2}{2!} + \cdots + \frac{q^n}{n!} + c_1 \cdot q^{n+1} \\ f_2(q) &= 1 + q + \frac{q^2}{2!} + \cdots + \frac{q^n}{n!} + c_2 \cdot q^{n+1} \end{aligned}$$

where $c_2 \neq c_1$.

Show that it is always possible to use blending:

$$\mathbf{x}^{\text{blended}} = \vartheta \cdot \mathbf{x}^1 + (1-\vartheta) \cdot \mathbf{x}^2$$

where $\mathbf{x}^1$ is the solution found using method $f_1(q)$ and $\mathbf{x}^2$ is the solution found using method $f_2(q)$, such that $\mathbf{x}^{\text{blended}}$ is of order $(n+1)$.

Find a formula for ϑ that will make the blended algorithm accurate to the order $(n+1)$.

[H3.4] RK Order Increase by Blending II

$$\begin{aligned} f_{blended}(q) &= 1 + q + \frac{q^2}{2!} + \cdots + \frac{q^n}{n!} + (\vartheta \cdot c_1 + (1-\vartheta) \cdot c_2) \cdot q^{n+1} \\ &\overset{!}{=} 1 + q + \frac{q^2}{2!} + \cdots + \frac{q^n}{n!} + \frac{q^{n+1}}{(n+1)!} \end{aligned}$$

$$\Rightarrow \quad \vartheta \cdot c_1 + (1-\vartheta) \cdot c_2 \overset{!}{=} \frac{1}{(n+1)!}$$

$$\Rightarrow \quad \vartheta = \frac{1 - c_2 \cdot (n+1)!}{(c_1 - c_2) \cdot (n+1)!}$$

has a solution for $c_2 \neq c_1$.

[H3.6] Runge-Kutta Integration

Given the following linear time-invariant continuous-time system:

$$\begin{aligned} \dot{\mathbf{x}} &= \begin{pmatrix} 1250 & -25113 & -60050 & -42647 & -23999 \\ 500 & -10068 & -24057 & -17092 & -9613 \\ 250 & -5060 & -12079 & -8586 & -4826 \\ -750 & 15101 & 36086 & 25637 & 14420 \\ 250 & -4963 & -11896 & -8438 & -4756 \end{pmatrix} \cdot \mathbf{x} + \begin{pmatrix} 5 \\ 2 \\ 1 \\ -3 \\ 1 \end{pmatrix} \cdot u \\ \mathbf{y} &= \begin{pmatrix} -1 & 26 & 59 & 43 & 23 \end{pmatrix} \cdot \mathbf{x} \end{aligned}$$

with initial conditions:

$$\mathbf{x_0} = \begin{pmatrix} 1 \\ -2 \\ 3 \\ -4 \\ 5 \end{pmatrix}$$

[H3.6] Runge-Kutta Integration II

Simulate the system across 10 seconds of simulated time with step input using the RK4 algorithm with the α-vector and β-matrix:

$$\alpha = \begin{pmatrix} 1/2 \\ 1/2 \\ 1 \\ 1 \end{pmatrix} \; ; \; \beta = \begin{pmatrix} 1/2 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1/6 & 1/3 & 1/3 & 1/6 \end{pmatrix}$$

The following fixed step sizes should be tried:

1. $h = 0.32$,
2. $h = 0.032$,
3. $h = 0.0032$.

Plot the three trajectories on top of each other. What can you conclude about the accuracy of the results?

[H3.6] Runge-Kutta Integration III

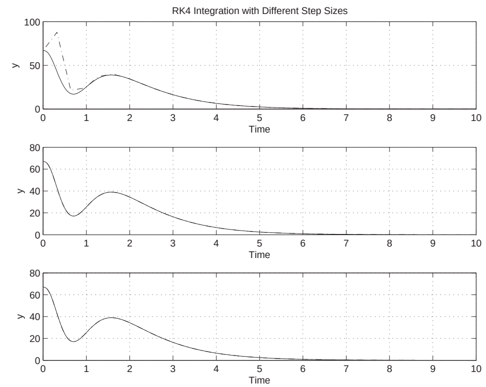

Figure: RK4 simulation with different step sizes

[H3.6] Runge-Kutta Integration IV

Defining the error as the infinity norm of the difference between the "correct" solution (computed using the lsim function) and the solution obtained using RK4, we find:

1. $err(h = 0.32) = 44.9013$,
2. $err(h = 0.032) = 0.0012$,
3. $err(h = 0.0032) = 1.1801e - 006$.

Clearly, $h = 0.32$ leads to an unacceptably large error (the difference between the two curves is clearly visible by naked eye), whereas the other two step sizes may be acceptable for most engineering problems.

[H3.12] BI4/$5_{0.45}$ Integration for Linear Systems

Given the following linear time-invariant continuous-time system:

$$\begin{aligned} \dot{\mathbf{x}} &= \begin{pmatrix} 1250 & -25113 & -60050 & -42647 & -23999 \\ 500 & -10068 & -24057 & -17092 & -9613 \\ 250 & -5060 & -12079 & -8586 & -4826 \\ -750 & 15101 & 36086 & 25637 & 14420 \\ 250 & -4963 & -11896 & -8438 & -4756 \end{pmatrix} \cdot \mathbf{x} + \begin{pmatrix} 5 \\ 2 \\ 1 \\ -3 \\ 1 \end{pmatrix} \cdot u \\ \mathbf{y} &= \begin{pmatrix} -1 & 26 & 59 & 43 & 23 \end{pmatrix} \cdot \mathbf{x} \end{aligned}$$

with initial conditions:

$$\mathbf{x_0} = \begin{pmatrix} 1 \\ -2 \\ 3 \\ -4 \\ 5 \end{pmatrix}$$

[H3.12] BI4/5$_{0.45}$ Integration for Linear Systems II

Simulate the system across 10 seconds of simulated time with step input using BI4/5$_{0.45}$. The explicit semi-step uses the fourth-order approximation of RKF4/5. There is no need to compute the fifth-order corrector. The implicit semi-step uses the fifth-order corrector. There is no need to compute the fourth–order corrector. Since the system to be simulated is linear, the implicit semi-step can be implemented using matrix inversion. No step-size control is attempted.

The following fixed step sizes should be tried:

1. $h = 0.32$,
2. $h = 0.032$,
3. $h = 0.0032$.

Plot the three trajectories on top of each other.

[H3.12] BI4/5$_{0.45}$ Integration for Linear Systems III

Let us find an integrator that can be used to simulate the linear problem:

$$\dot{\mathbf{x}} = \mathbf{A} \cdot \mathbf{x} + \mathbf{B} \cdot \mathbf{u}$$

assuming a zero-order hold (ZOH) on the input, i.e., $\mathbf{u} = \mathbf{u_k}, \forall t \in [t_k, t_{k+1}]$.

Let us demonstrate the approach for the RK4 algorithm introduced in class:

$$
\begin{array}{lrcl}
0^{\text{th}} \text{ stage:} & \dot{\mathbf{x}}_\mathbf{k} & = & \mathbf{f}(\mathbf{x_k}, t_k) \\
1^{\text{st}} \text{ stage:} & \mathbf{x}^{\mathbf{P}_1} & = & \mathbf{x_k} + \frac{h}{2} \cdot \dot{\mathbf{x}}_\mathbf{k} \\
 & \dot{\mathbf{x}}^{\mathbf{P}_1} & = & \mathbf{f}(\mathbf{x}^{\mathbf{P}_1}, t_{k+\frac{1}{2}}) \\
2^{\text{nd}} \text{ stage:} & \mathbf{x}^{\mathbf{P}_2} & = & \mathbf{x_k} + \frac{h}{2} \cdot \dot{\mathbf{x}}^{\mathbf{P}_1} \\
 & \dot{\mathbf{x}}^{\mathbf{P}_2} & = & \mathbf{f}(\mathbf{x}^{\mathbf{P}_2}, t_{k+\frac{1}{2}}) \\
3^{\text{rd}} \text{ stage:} & \mathbf{x}^{\mathbf{P}_3} & = & \mathbf{x_k} + h \cdot \dot{\mathbf{x}}^{\mathbf{P}_2} \\
 & \dot{\mathbf{x}}^{\mathbf{P}_3} & = & \mathbf{f}(\mathbf{x}^{\mathbf{P}_3}, t_{k+1}) \\
4^{\text{th}} \text{ stage:} & \mathbf{x_{k+1}} & = & \mathbf{x_k} + \frac{h}{6} \cdot [\dot{\mathbf{x}}_\mathbf{k} + 2 \cdot \dot{\mathbf{x}}^{\mathbf{P}_1} + 2 \cdot \dot{\mathbf{x}}^{\mathbf{P}_2} + \dot{\mathbf{x}}^{\mathbf{P}_3}]
\end{array}
$$

[H3.12] BI4/5$_{0.45}$ Integration for Linear Systems IV

$$
\begin{array}{rcl}
\dot{\mathbf{x}}_\mathbf{k} & = & \mathbf{A} \cdot \mathbf{x_k} + \mathbf{B} \cdot \mathbf{u_k} = \mathbf{A_0} \cdot \mathbf{x_k} + \mathbf{B_0} \cdot \mathbf{u_k} \\
\mathbf{x}^{\mathbf{P}_1} & = & \mathbf{x_k} + \frac{h}{2} \cdot (\mathbf{A_0} \cdot \mathbf{x_k} + \mathbf{B_0} \cdot \mathbf{u_k}) = \mathbf{F_1} \cdot \mathbf{x_k} + \mathbf{G_1} \cdot \mathbf{u_k} \\
\dot{\mathbf{x}}^{\mathbf{P}_1} & = & \mathbf{A} \cdot (\mathbf{F_1} \cdot \mathbf{x_k} + \mathbf{G_1} \cdot \mathbf{u_k}) + \mathbf{B} \cdot \mathbf{u}_{\mathbf{k}+\frac{1}{2}} = \mathbf{A_1} \cdot \mathbf{x_k} + \mathbf{B_1} \cdot \mathbf{u_k} \\
\mathbf{x}^{\mathbf{P}_2} & = & \mathbf{x_k} + \frac{h}{2} \cdot (\mathbf{A_1} \cdot \mathbf{x_k} + \mathbf{B_1} \cdot \mathbf{u_k}) = \mathbf{F_2} \cdot \mathbf{x_k} + \mathbf{G_2} \cdot \mathbf{u_k} \\
\dot{\mathbf{x}}^{\mathbf{P}_2} & = & \mathbf{A} \cdot (\mathbf{F_2} \cdot \mathbf{x_k} + \mathbf{G_2} \cdot \mathbf{u_k}) + \mathbf{B} \cdot \mathbf{u}_{\mathbf{k}+\frac{1}{2}} = \mathbf{A_2} \cdot \mathbf{x_k} + \mathbf{B_2} \cdot \mathbf{u_k} \\
\mathbf{x}^{\mathbf{P}_3} & = & \mathbf{x_k} + h \cdot (\mathbf{A_2} \cdot \mathbf{x_k} + \mathbf{B_2} \cdot \mathbf{u_k}) = \mathbf{F_3} \cdot \mathbf{x_k} + \mathbf{G_3} \cdot \mathbf{u_k} \\
\dot{\mathbf{x}}^{\mathbf{P}_3} & = & \mathbf{A} \cdot (\mathbf{F_3} \cdot \mathbf{x_k} + \mathbf{G_3} \cdot \mathbf{u_k}) + \mathbf{B} \cdot \mathbf{u_{k+1}} = \mathbf{A_3} \cdot \mathbf{x_k} + \mathbf{B_3} \cdot \mathbf{u_k} \\
\mathbf{x_{k+1}} & = & \mathbf{x_k} + \frac{h}{6} \cdot (\mathbf{A_0} \cdot \mathbf{x_k} + \mathbf{B_0} \cdot \mathbf{u_k} + 2 \cdot (\mathbf{A_1} \cdot \mathbf{x_k} + \mathbf{B_1} \cdot \mathbf{u_k}) + 2 \cdot (\mathbf{A_2} \cdot \mathbf{x_k} + \mathbf{B_2} \cdot \mathbf{u_k}) + \mathbf{A_3} \cdot \mathbf{x_k} + \mathbf{B_3} \cdot \mathbf{u_k}) \\
 & = & \mathbf{F} \cdot \mathbf{x_k} + \mathbf{G} \cdot \mathbf{u_k}
\end{array}
$$

We can apply the same technique to Runge-Kutta-Fehlberg:

$[F4, G4, F5, G5] = \text{rkf45\_lin}(A, B, h);$

[H3.12] BI4/5$_{0.45}$ Integration for Linear Systems V

We can now implement the linear $BI4/5_{0.45}$ algorithm. The forward semi-step is:

$$\mathbf{x}_{\mathbf{k}+\frac{1}{2}} = \mathbf{F_4} \cdot \mathbf{x_k} + \mathbf{G_4} \cdot \mathbf{u_k}$$

and the backward semi-step can be written as:

$$\mathbf{x}_{\mathbf{k}+\frac{1}{2}} = \mathbf{F_5} \cdot \mathbf{x_{k+1}} + \mathbf{G_5} \cdot \mathbf{u_{k+1}}$$

and since we assume $\mathbf{u_{k+1}} = \mathbf{u_k}$:

$$\mathbf{x_{k+1}} = \mathbf{F_5}^{-1} \cdot \mathbf{x}_{\mathbf{k}+\frac{1}{2}} - \mathbf{F_5}^{-1} \cdot \mathbf{G_5} \cdot \mathbf{u_k}$$

Therefore:

$$\mathbf{x_{k+1}} = \mathbf{F_5}^{-1} \cdot \mathbf{F_4} \cdot \mathbf{x_k} + \mathbf{F_5}^{-1} \cdot (\mathbf{G_4} - \mathbf{G_5}) \cdot \mathbf{u_k} = \mathbf{F_{BI}} \cdot \mathbf{x_k} + \mathbf{G_{BI}} \cdot \mathbf{u_k}$$

[H3.12] $BI4/5_{0.45}$ Integration for Linear Systems VI

Implemented:

```
function [F, G] = bi45t_lin(A, B, h, theta)
    [F4, G4, dummy1, dummy2] = rkf45_lin(A, B, theta * h);
    [dummy1, dummy2, F5, G5] = rkf45_lin(A, B, (theta - 1) * h);
    F = F5\F4;
    G = F5\(G4 - G5);
return
```

Now the simulation is a simple loop.

[H3.12] $BI4/5_{0.45}$ Integration for Linear Systems VII

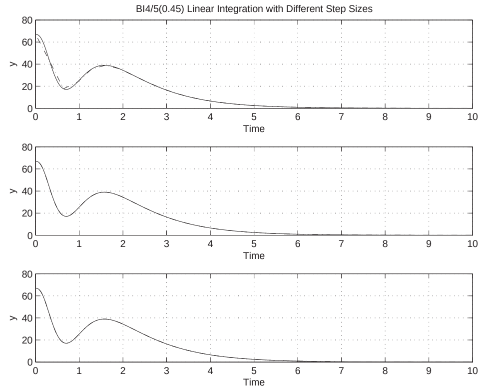

Figure: $BI4/5_{0.45}$ linear simulation with different step sizes

[H3.12] $BI4/5_{0.45}$ Integration for Linear Systems VIII

Defining the error as the infinity norm of the difference between the "correct" solution (computed using the lsim function) and the solution obtained using the linear version of $BI4/5_{0.45}$, we find:

1. $err(h = 0.32) = 0.0912$,
2. $err(h = 0.032) = 3.4369e - 006$,
3. $err(h = 0.0032) = 2.0777e - 006$.

The solution with $h = 0.32$ leads to an error that is a bit on the large side but still quite reasonable, whereas the solution with $h = 0.032$ is perfect. Using $h = 0.0032$, we don't gain much. The roundoff (shift-out) error kills the additional accuracy that we might gain when using a larger mantissa.

[H3.14] $BI4/5_{0.45}$ Integration for Non-linear Systems

Repeat Hw.[H3.12]. This time, we want to replace the matrix inversion by Newton iteration. Of course, since the problem is linear and time-invariant, Newton iteration and modified Newton iteration are identical. Iterate until $\delta_{\mathrm{rel}} \leq 10^{-5}$, where:

$$\delta_{\mathrm{rel}} = \frac{\|\mathbf{x}^{\mathrm{right}}_{k+\frac{1}{2}} - \mathbf{x}^{\mathrm{left}}_{k+\frac{1}{2}}\|_\infty}{\max(\|\mathbf{x}^{\mathrm{left}}_{k+\frac{1}{2}}\|_2, \|\mathbf{x}^{\mathrm{right}}_{k+\frac{1}{2}}\|_2, \delta)}$$

Compare the results obtained with those found in Hw.[H3.12].

[H3.14] BI4/$5_{0.45}$ Integration for Non-linear Systems II

We implement the Newton iteration in the routine that computes a single step of $BI4/5_{0.45}$:

```
function [xnew] = bi45t_step(x, t, h, theta, tol)
    H = hessian(x, t, (theta − 1) ∗ h, 4);
    [x_left, dummy] = rkf45_step(x, t, theta ∗ h);
    xnew = x_left;
    err = 1;
    while err > 0.1 ∗ tol,
        [dummy, x_right] = rkf45_step(xnew, t + h, (theta − 1) ∗ h);
        xnew = xnew − H\(x_right − x_left);
        err = norm(x_right − x_left,'inf')/max([norm(x_left),norm(x_right), 1.0e − 10]);
    end
return
```


[H3.14] BI4/$5_{0.45}$ Integration for Non-linear Systems III

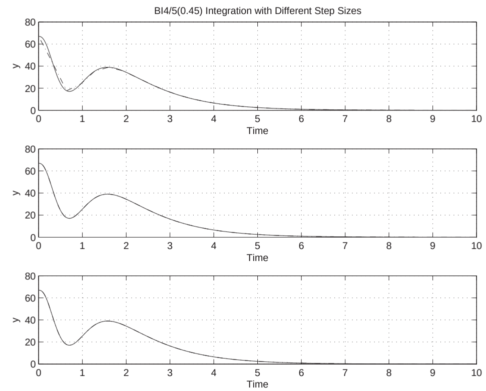

Figure: $BI4/5_{0.45}$ non-linear simulation with different step sizes

[H3.14] BI4/$5_{0.45}$ Integration for Non-linear Systems IV

Defining the error as the infinity norm of the difference between the "correct" solution (computed using the lsim function) and the solution obtained using the non-linear version of $BI4/5_{0.45}$, we find:

1. $err(h = 0.32) = 0.0912$,
2. $err(h = 0.032) = 3.4248e - 006$,
3. $err(h = 0.0032) = 1.2194e - 006$.

The results are almost identical as for the linear solution. This is not surprising, because for a linear system, Newton iteration converges to the correct solution within a single iteration step.

[H3.14] BI4/$5_{0.45}$ Integration for Non-linear Systems V

Building a special linear version of the $BI4/5_{0.45}$ code wasn't worth it, and is hardly ever done. In fact, we were lucky, because, by simulating a step response, the ZOH applied to the input had no influence on the solution. In general, the non-linear solution will be better, because it makes use of the correct values of the inputs throughout the step.

The only situation, where we might want to use the linear version is in a *real-time context*. The linear $BI4/5_{0.45}$ code computes almost everything off-line, i.e., before the simulation starts. The simulation loop is reduced to two multiplications and an addition. The non-linear $BI4/5_{0.45}$ code needs to call bi45t_step, which in return calls rkf45_step twice, during each simulation step.