5

Second Derivative Systems

Preview

In this chapter, we shall look at integration algorithms designed to deal with system descriptions containing second–order derivatives in time. Such system descriptions occur naturally in the mathematical modeling of mechanical systems, as well as in the mathematical modeling of distributed parameter systems leading to hyperbolic partial differential equations.

In this chapter, we shall concentrate on mechanical systems. The discussion of partial differential equations is postponed to the next chapter.

Whereas it is always possible to convert second derivative systems to state–space form, integration algorithms that deal with the second derivatives directly may, in some cases, offer a numerical advantage.

5.1 Introduction

Let us start the discussion by modeling a human body riding in a car. Some people with weak muscles in their neck region suffer from a so–called *cervical syndrome* [5.8]. When riding in a car for extended periods of time, they suffer awful headache attacks, because their head vibrates (oscillates) with the vibrations of the car on the road, since their head is not attached stiffly enough to the shoulders.

Since the passengers in a car are seated, their legs can be eliminated from the model, as they won't affect the motion of the neck at all. The body can be modeled in a first approximation as a mechanical system. A very simple mechanical model of a sitting human body is depicted in Fig.5.1.

The model is decomposed into four masses that can move separately from each other, representing the head, the upper torso, the two arms, and the lower body. Due to linearity, the two arms can be treated as a single mass. The connections between the four masses are modeled as damped springs. The bumping of the road is modeled by a time–dependent force attached to the lower body. To be determined is the distance between head and shoulders as a function of the driving force.

Only vertical motions are to be considered. Hence each mass represents a single mechanical degree of freedom. The overall model is obtained by applying Newton's law to each of the four masses separately:

$$M_1 \cdot \ddot{x}_1 \quad = \quad k_1 \cdot (x_2 - x_1) + B_1 \cdot (\dot{x}_2 - \dot{x}_1) \tag{5.1}$$

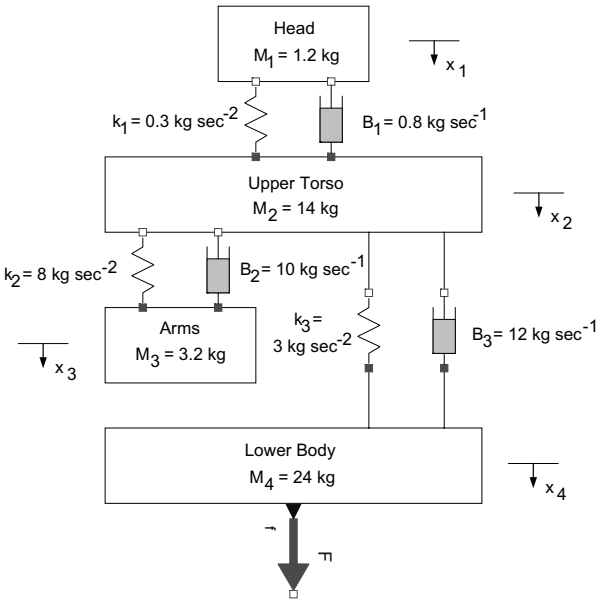

FIGURE 5.1. Mechanical model of a sitting human body.

$$\begin{aligned} M_2 \cdot \ddot{x}_2 &= k_2 \cdot (x_3 - x_2) + B_2 \cdot (\dot{x}_3 - \dot{x}_2) + k_3 \cdot (x_4 - x_2) \\ &\quad + B_3 \cdot (\dot{x}_4 - \dot{x}_2) - k_1 \cdot (x_2 - x_1) - B_1 \cdot (\dot{x}_2 - \dot{x}_1) \quad (5.2) \\ M_3 \cdot \ddot{x}_3 &= -k_2 \cdot (x_3 - x_2) - B_2 \cdot (\dot{x}_3 - \dot{x}_2) \quad (5.3) \\ M_4 \cdot \ddot{x}_4 &= F - k_3 \cdot (x_4 - x_2) - B_3 \cdot (\dot{x}_4 - \dot{x}_2) \quad (5.4) \end{aligned}$$

Let:

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} \quad (5.5)$$

be the partial state vector consisting of the four mass positions. Using the partial state vector, the model can be written in matrix/vector form as follows:

$$\mathbf{M} \cdot \ddot{\mathbf{x}} + \mathbf{C} \cdot \dot{\mathbf{x}} + \mathbf{K} \cdot \mathbf{x} = \mathbf{f} \quad (5.6)$$

where:

$$\mathbf{M} = \begin{pmatrix} M_1 & 0 & 0 & 0 \\ 0 & M_2 & 0 & 0 \\ 0 & 0 & M_3 & 0 \\ 0 & 0 & 0 & M_4 \end{pmatrix} \quad (5.7)$$

is the *mass matrix*,

$$\mathbf{C} = \begin{pmatrix} B_1 & -B_1 & 0 & 0 \\ -B_1 & (B_1 + B_2 + B_3) & 0 & 0 \\ 0 & -B_2 & B_2 & 0 \\ 0 & -B_3 & 0 & B_3 \end{pmatrix} \tag{5.8}$$

is the *damping matrix*,

$$\mathbf{K} = \begin{pmatrix} k_1 & -k_1 & 0 & 0 \\ -k_1 & (k_1 + k_2 + k_3) & 0 & 0 \\ 0 & -k_2 & k_2 & 0 \\ 0 & -k_3 & 0 & k_3 \end{pmatrix} \tag{5.9}$$

is the *stiffness matrix*, and

$$\mathbf{f} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ F \end{pmatrix} \tag{5.10}$$

is the vector of (generalized) forces.

The mass matrix turned out to be a diagonal matrix in this example, but this is only true, because no rotational motions were considered in the given example. Generally, this will not be the case.

Assuming that the mass matrix is non–singular, i.e., there are as many mechanical degrees of freedom in the system as were formulated into second–order differential equations, i.e., there are no *structural singularities* in the model [5.3], the model can be solved for the highest derivatives:

$$\ddot{\mathbf{x}} = \mathbf{A}^2 \cdot \mathbf{x} + \mathbf{B} \cdot \dot{\mathbf{x}} + \mathbf{u} \tag{5.11}$$

where:

$$\mathbf{A} = \sqrt{-\mathbf{M}^{-1} \cdot \mathbf{K}} \tag{5.12}$$

$$\mathbf{B} = -\mathbf{M}^{-1} \cdot \mathbf{C} \tag{5.13}$$

$$\mathbf{u} = \mathbf{M}^{-1} \cdot \mathbf{f} \tag{5.14}$$

or, more generally, in a nonlinear case:

$$\ddot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \dot{\mathbf{x}}, \mathbf{u}, t) \tag{5.15}$$

Of special interest is the case of the *conservative*, i.e., friction–less system with the second–derivative form:

$$\ddot{\mathbf{x}} = \mathbf{A}^2 \cdot \mathbf{x} + \mathbf{u} \tag{5.16}$$

or, more generally:

$$\ddot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, t) \tag{5.17}$$

and especially, we may want to look at the homogeneous, conservative, linear system with the second–derivative model:

$$\ddot{\mathbf{x}} = \mathbf{A}^2 \cdot \mathbf{x} \tag{5.18}$$

5.2 Conversion of Second–derivative Models to State–space Form

It is always possible to convert a second–derivative model to state–space form. To this end, we introduce the velocity vector, $\mathbf{v}$:

$$\dot{\mathbf{x}} = \mathbf{v} \tag{5.19}$$

$$\dot{\mathbf{v}} = \ddot{\mathbf{x}} = \mathbf{A}^2 \cdot \mathbf{x} + \mathbf{B} \cdot \mathbf{v} + \mathbf{u} \tag{5.20}$$

We introduce the state vector:

$$\xi = \begin{pmatrix} \mathbf{x} \\ \mathbf{v} \end{pmatrix} \tag{5.21}$$

which leads us to the state–space form:

$$\dot{\xi} = \begin{pmatrix} \mathbf{Z}^{(n)} & \mathbf{I}^{(n)} \\ \mathbf{A}^2 & \mathbf{B} \end{pmatrix} \cdot \xi + \begin{pmatrix} \mathbf{0}^{(n)} \\ \mathbf{u} \end{pmatrix} \tag{5.22}$$

where $\mathbf{Z}^{(n)}$ is a zero matrix of dimensions $n \times n$, $\mathbf{I}^{(n)}$ is an identity matrix of dimensions $n \times n$, and $\mathbf{0}^{(n)}$ is a zero vector of length n.

After the conversion, the state–space model can be simulated using any of the integration algorithms introduced in the previous chapters of this book.

Unfortunately, the resulting state vector is of length $2 \cdot n$, which makes simulation methods (integration algorithms) that can deal with the second–derivative model directly potentially interesting.

We might conclude at this point in time that direct methods should be of particular interest for the simulation of *conservative systems*, as those systems do not require computing the velocity vector at all, i.e., half of the state variables can simply be thrown away.

5.3 Velocity–free Models

We shall define a velocity–free model as one that satisfies, in the linear case, the differential vector equation:

$$\ddot{\mathbf{x}} = \mathbf{A}^2 \cdot \mathbf{x} + \mathbf{u} \tag{5.23}$$

and, in the nonlinear case, the differential vector equation:

$$\ddot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, t) \tag{5.24}$$

Notice that every conservative system leads to a velocity–free second–derivative model. Yet, not every velocity–free second–derivative model is conservative. This can be recognized easily.

Given a linear, time–invariant, homogeneous state–space model of the form:

$$\dot{\mathbf{x}} = \mathbf{A} \cdot \mathbf{x} \tag{5.25}$$

Depending on the eigenvalues of $\mathbf{A}$, the system is either damped or undamped, stable or unstable. We can differentiate the state–space model, leading to:

$$\ddot{\mathbf{x}} = \mathbf{A} \cdot \dot{\mathbf{x}} = \mathbf{A}^2 \cdot \mathbf{x} \tag{5.26}$$

Thus, any linear, time–invariant, homogeneous state–space model can also be written in the form of a velocity–free second–derivative model, irrespective of where its eigenvalues are located. Yet, a conservative linear system has its eigenvalues spread up and down along the imaginary axis of the complex plane.

How can the special structure of a velocity–free second–derivative model be exploited by a simulation algorithm?

Let us start by developing the solution vector at time $(t+h)$ into a Taylor series around time t:

$$\mathbf{x_{k+1}} = \mathbf{x_k} + h \cdot \dot{\mathbf{x}}_\mathbf{k} + \frac{h^2}{2} \cdot \ddot{\mathbf{x}}_\mathbf{k} + \frac{h^3}{6} \cdot \mathbf{x_k}^{(iii)} + \frac{h^4}{24} \cdot \mathbf{x_k}^{(iv)} + \dots \tag{5.27}$$

We shall also need to develop the solution vector at time $(t-h)$ into a Taylor series around time t:

$$\mathbf{x_{k-1}} = \mathbf{x_k} - h \cdot \dot{\mathbf{x}}_\mathbf{k} + \frac{h^2}{2} \cdot \ddot{\mathbf{x}}_\mathbf{k} - \frac{h^3}{6} \cdot \mathbf{x_k}^{(iii)} + \frac{h^4}{24} \cdot \mathbf{x_k}^{(iv)} \mp \dots \tag{5.28}$$

Adding these two equations together, we obtain:

$$\mathbf{x_{k+1}} + \mathbf{x_{k-1}} = 2 \cdot \mathbf{x_k} + h^2 \cdot \ddot{\mathbf{x}}_\mathbf{k} + \frac{h^4}{12} \cdot \mathbf{x_k}^{(iv)} + \dots \tag{5.29}$$

thus:

$$\mathbf{x_{k+1}} = 2 \cdot \mathbf{x_k} - \mathbf{x_{k-1}} + h^2 \cdot \ddot{\mathbf{x}}_\mathbf{k} + \mathrm{o}(h^4) \tag{5.30}$$

We just found a 3rd–order accurate explicit linear multi–step method that makes use of the second derivative directly. In some references, the method is referred to as Godunov's method [5.2], in spite of the fact that Sergei Konstantinovic Godunov, a famous Russian applied mathematician of the middle of the 20th century, was much more interested in *conservation laws*, i.e., in partial differential equations of the hyperbolic type, where he used this technique to approximate spatial derivatives.

The second derivative can be plugged in from the homogeneous linear second–derivative model of Eq.(5.18):

$$\mathbf{x_{k+1}} \approx 2 \cdot \mathbf{x_k} - \mathbf{x_{k-1}} + (\mathbf{A} \cdot h)^2 \cdot \mathbf{x_k} \tag{5.31}$$

We find the $\mathbf{F}$–matrix of the algorithm in the usual fashion. Let

$$\xi_{\mathbf{k}} = \begin{pmatrix} \mathbf{x_{k-1}} \\ \mathbf{x_k} \end{pmatrix} \tag{5.32}$$

Then:

$$\xi_{\mathbf{k+1}} \approx \mathbf{F} \cdot \xi_{\mathbf{k}} \tag{5.33}$$

where:

$$\mathbf{F} = \begin{pmatrix} \mathbf{Z}^{(n)} & \mathbf{I}^{(n)} \\ -\mathbf{I}^{(n)} & \left[2 \cdot \mathbf{I}^{(n)} + (\mathbf{A} \cdot h)^2\right] \end{pmatrix} \tag{5.34}$$

Consequently, we should be able to plot the stability domain of this algorithm as a function of the eigenvalues of $\mathbf{A} \cdot h$, exactly as we did in the case of the algorithms presented in the previous two chapters. We shall attempt to do so in due course.

5.4 Linear Velocity Models

We shall next look at the case of a possibly nonlinear second derivative model with a linear velocity term, i.e., a model of the type:

$$\ddot{\mathbf{x}} + \mathbf{B} \cdot \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, t) \tag{5.35}$$

We shall demonstrate that this problem can be reduced to the case of the velocity–free model.

To this end, we apply the variable transformation:

$$\xi = \exp\left(\frac{\mathbf{B} \cdot t}{2}\right) \cdot \mathbf{x} \tag{5.36}$$

Therefore:

$$\mathbf{x} = \exp\left(\frac{-\mathbf{B}\cdot t}{2}\right)\cdot\xi \tag{5.37}$$

$$\dot{\mathbf{x}} = -\frac{\mathbf{B}}{2}\cdot\exp\left(\frac{-\mathbf{B}\cdot t}{2}\right)\cdot\xi + \exp\left(\frac{-\mathbf{B}\cdot t}{2}\right)\cdot\dot{\xi} \tag{5.38}$$

$$\ddot{\mathbf{x}} = \frac{\mathbf{B}^2}{4}\cdot\exp\left(\frac{-\mathbf{B}\cdot t}{2}\right)\cdot\xi - \mathbf{B}\cdot\exp\left(\frac{-\mathbf{B}\cdot t}{2}\right)\cdot\dot{\xi} + \exp\left(\frac{-\mathbf{B}\cdot t}{2}\right)\cdot\ddot{\xi} \tag{5.39}$$

We introduce the abbreviation:

$$\mathbf{E} = \exp\left(\frac{-\mathbf{B}\cdot t}{2}\right) \tag{5.40}$$

Thus:

$$\mathbf{x} = \mathbf{E}\cdot\xi \tag{5.41}$$

$$\dot{\mathbf{x}} = -\frac{\mathbf{B}}{2}\cdot\mathbf{E}\cdot\xi + \mathbf{E}\cdot\dot{\xi} \tag{5.42}$$

$$\ddot{\mathbf{x}} = \frac{\mathbf{B}^2}{4}\cdot\mathbf{E}\cdot\xi - \mathbf{B}\cdot\mathbf{E}\cdot\dot{\xi} + \mathbf{E}\cdot\ddot{\xi} \tag{5.43}$$

Plugging these expressions into the original second–derivative model, we obtain:

$$\ddot{\xi} = \mathbf{E}^{-1}\cdot\frac{\mathbf{B}^2}{4}\cdot\mathbf{E}\cdot\xi + \mathbf{E}^{-1}\cdot\mathbf{f}(\mathbf{E}\cdot\xi,\mathbf{u},t) \tag{5.44}$$

which has taken on the form of a velocity–free second–derivative model.

5.5 Nonlinear Velocity Models

If the second–derivative model assumes the general form:

$$\ddot{\mathbf{x}} = \mathbf{f}(\mathbf{x},\dot{\mathbf{x}},\mathbf{u},t) \tag{5.45}$$

the velocity vector $\dot{\mathbf{x}}$ needs to be computed as well. However, since we have access to the second derivative, we can compute the velocity vector using any one of the integration algorithms proposed in the previous two chapters.

It seems reasonable to employ an explicit linear multi–step algorithm. Furthermore, since the derivative vector always gets multiplied by the step

size, h, it suffices to use a second–order accurate formula. A reasonable choice might be AB2:

$$\dot{\mathbf{x}}_{\mathbf{k+1}} = \dot{\mathbf{x}}_{\mathbf{k}} + \frac{h}{2} \cdot (3 \cdot \ddot{\mathbf{x}}_{\mathbf{k}} - \ddot{\mathbf{x}}_{\mathbf{k-1}}) \tag{5.46}$$

For the computation of the stability domain, we are dealing with the following set of three equations:

$$\begin{aligned} \mathbf{x}_{\mathbf{k+1}} &= 2 \cdot \mathbf{x}_{\mathbf{k}} - \mathbf{x}_{\mathbf{k-1}} + h^2 \cdot \dot{\mathbf{v}}_{\mathbf{k}} && (5.47) \\ \mathbf{v}_{\mathbf{k+1}} &= \mathbf{v}_{\mathbf{k}} + \frac{h}{2} \cdot (3 \cdot \dot{\mathbf{v}}_{\mathbf{k}} - \dot{\mathbf{v}}_{\mathbf{k-1}}) && (5.48) \\ \dot{\mathbf{v}}_{\mathbf{k}} &= \mathbf{A}^2 \cdot \mathbf{x}_{\mathbf{k}} + \mathbf{B} \cdot \mathbf{v}_{\mathbf{k}} && (5.49) \end{aligned}$$

Plugging the linear second–derivative model of Eq.(5.49) into the two integrator equations, Eq.(5.47) and Eq.(5.48), we obtain:

$$\begin{aligned} \mathbf{x}_{\mathbf{k+1}} &= 2 \cdot \mathbf{x}_{\mathbf{k}} - \mathbf{x}_{\mathbf{k-1}} + (\mathbf{A} \cdot h)^2 \cdot \mathbf{x}_{\mathbf{k}} + (\mathbf{B} \cdot h) \cdot (h \cdot \mathbf{v}_{\mathbf{k}}) && (5.50) \\ (h \cdot \mathbf{v}_{\mathbf{k+1}}) &= (h \cdot \mathbf{v}_{\mathbf{k}}) + \frac{3}{2} \cdot (\mathbf{A} \cdot h)^2 \cdot \mathbf{x}_{\mathbf{k}} + \frac{3}{2} \cdot (\mathbf{B} \cdot h) \cdot (h \cdot \mathbf{v}_{\mathbf{k}}) \\ &\quad - \frac{1}{2} \cdot (\mathbf{A} \cdot h)^2 \cdot \mathbf{x}_{\mathbf{k-1}} - \frac{1}{2} \cdot (\mathbf{B} \cdot h) \cdot (h \cdot \mathbf{v}_{\mathbf{k-1}}) && (5.51) \end{aligned}$$

which can be rewritten in a matrix/vector form as follows:

$$\begin{pmatrix} \mathbf{x}_{\mathbf{k}} \\ h \cdot \mathbf{v}_{\mathbf{k}} \\ \mathbf{x}_{\mathbf{k+1}} \\ h \cdot \mathbf{v}_{\mathbf{k+1}} \end{pmatrix} = \mathbf{F} \cdot \begin{pmatrix} \mathbf{x}_{\mathbf{k-1}} \\ h \cdot \mathbf{v}_{\mathbf{k-1}} \\ \mathbf{x}_{\mathbf{k}} \\ h \cdot \mathbf{v}_{\mathbf{k}} \end{pmatrix} \tag{5.52}$$

where:

$$\mathbf{F} = \begin{pmatrix} \mathbf{Z}^{(n)} & \mathbf{Z}^{(n)} & \mathbf{I}^{(n)} & \mathbf{Z}^{(n)} \\ \mathbf{Z}^{(n)} & \mathbf{Z}^{(n)} & \mathbf{Z}^{(n)} & \mathbf{I}^{(n)} \\ -\mathbf{I}^{(n)} & \mathbf{Z}^{(n)} & \left[2 \cdot \mathbf{I}^{(n)} + (\mathbf{A} \cdot h)^2\right] & \mathbf{B} \cdot h \\ -\frac{1}{2} \cdot (\mathbf{A} \cdot h)^2 & -\frac{1}{2} \cdot (\mathbf{B} \cdot h) & \frac{3}{2} \cdot (\mathbf{A} \cdot h)^2 & \left[\mathbf{I}^{(n)} + \frac{3}{2} \cdot (\mathbf{B} \cdot h)\right] \end{pmatrix} \tag{5.53}$$

5.6 Stability and Damping of Godunov Scheme

As with all other integration techniques, we shall now introduce a classification code for the Godunov scheme. Since the algorithm is explicit and third–order accurate, we shall call this scheme GE3. We shall call the enhanced scheme, that also computes the velocity vector, GE3/AB2.

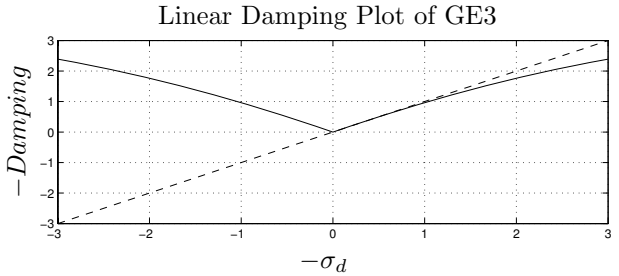

FIGURE 5.2. Linear damping plot of GE3 algorithm.

Before we attempt drawing a stability domain of GE3, we shall draw the linear damping plot. It is shown in Fig. 5.2.

How very disappointing! The scheme is unstable in the left half plane. The result should not surprise us too much. Since the **F**–matrix of Eq.(5.34) is an even function in $\mathbf{A} \cdot h$, the damping properties must be symmetric to the imaginary axis. Thus there cannot exist an asymptotic region around the origin, as we would expect of any well–behaved integration algorithm.

To gain a better understanding of the damping properties of the algorithm, let us plot the damping order star.

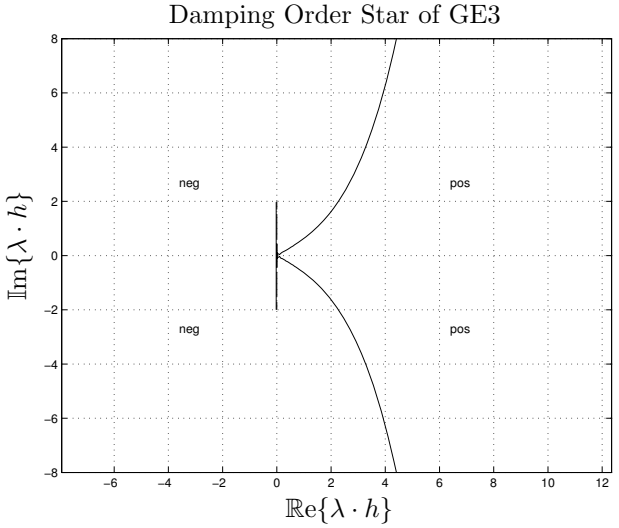

FIGURE 5.3. Damping order star of GE3 algorithm.

Interesting is the line segment stretching from $-2j$ to $+2j$ along the imaginary axis. Evidently, there is zero damping along this line segment, which is exactly, what it should be. To verify the results, let us plot the linear damping properties once more, but this time along the imaginary rather than the real axis.

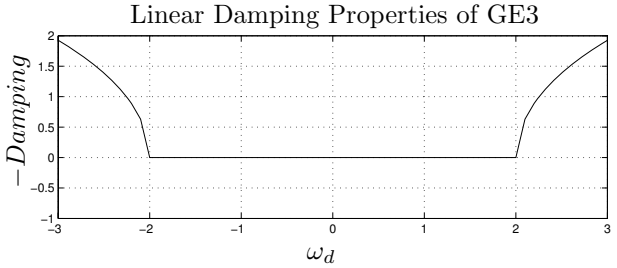

FIGURE 5.4. Linear damping properties of GE3 along imaginary axis.

The GE3 algorithm is only useful for strictly conservative systems. In order to obtain marginally stable results, the largest absolute eigenvalue multiplied by the step size must be smaller than or equal to 2:

$$|\lambda|_{\max} \cdot h \leq 2 \tag{5.54}$$

Let us now plot the linear frequency properties of GE3 along the imaginary axis.

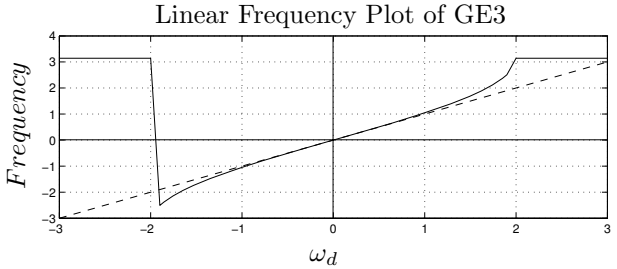

FIGURE 5.5. Linear frequency plot of GE3 algorithm.

The algorithm produces results that are decently accurate, if

$$|\lambda|_{\max} \cdot h \leq 1 \tag{5.55}$$

Let us now discuss the stability and damping properties of the GE3/AB2 algorithm, which is characterized by the **F**–matrix of Eq.(5.53).

When plotting the stability domain, the elements of the **B**–matrix cannot be chosen independently of those of the **A**–matrix. They must be chosen such that the overall system has its eigenvalues located on the unit circle. The procedure is explained in detail in Hw.[H5.3].

Since this is a third–order accurate linear explicit multi–step method similar in scope to AB3, we decided to plot the stability domain of AB3 on top of the stability domain of GE3/AB2.

GE3/AB2 performs similar to AB3 for systems with eigenvalues located in the vicinity of the negative real axis, but it outperforms AB3 when

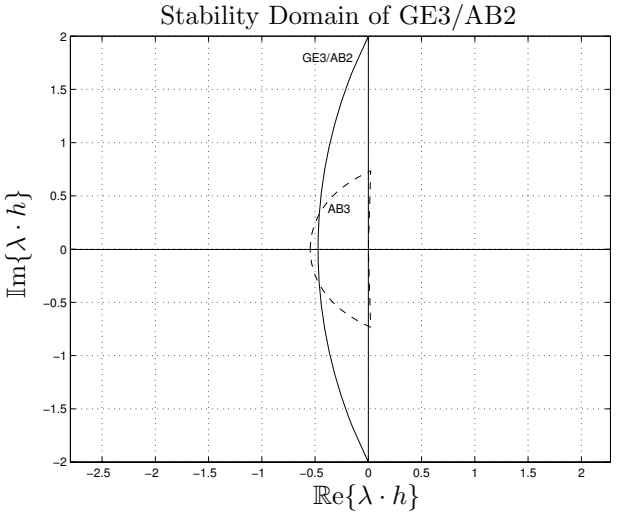

FIGURE 5.6. Stability domain of GE3/AB2 algorithm.

employed to simulating systems with their eigenvalues located close to the imaginary axis.

5.7 Explicit and Implicit Godunov Algorithms of Different Orders

Classes of both explicit and implicit integration algorithms of different orders of approximation accuracy for second–derivative systems can be derived using Newton–Gregory polynomials.

To this end, we shall develop $\mathbf{x}(t)$ into a Newton–Gregory backward polynomial around t_{k+1}. We then compute the second derivative of the Newton–Gregory polynomial. Evaluating this second derivative polynomial for $s = -1$, we obtain the class of explicit Godunov schemes. Evaluating the second derivative polynomial for $s = 0$, we obtain the class of implicit Godunov methods.

We shall denote the explicit Godunov scheme of order n as GE_n, and the implicit Godunov algorithm of the same order as GI_n. The enhanced algorithms, that also compute the velocity vector, are denoted as $\mathrm{GE}_n/\mathrm{AB}_{n-1}$ and $\mathrm{GI}_n/\mathrm{BDF}_{n-1}$, respectively.

Since the approach is very similar to those presented in Chapter 4 for the derivation of the AB, AM, and BDF algorithms, we shall refrain from repeating this derivation here once more. After all, some problems need to remain to be dealt with in the homework section.

The resulting algorithms are summarized below:

$$GE3: \quad \mathbf{x_{k+1}} = 2 \cdot \mathbf{x_k} - \mathbf{x_{k-1}} + h^2 \cdot \ddot{\mathbf{x}}_\mathbf{k} \tag{5.56}$$

$$GE4: \quad \mathbf{x_{k+1}} = \frac{20}{11} \cdot \mathbf{x_k} - \frac{6}{11} \cdot \mathbf{x_{k-1}} - \frac{4}{11} \cdot \mathbf{x_{k-2}} + \frac{1}{11} \cdot \mathbf{x_{k-3}} + \frac{12}{11} \cdot h^2 \cdot \ddot{\mathbf{x}}_\mathbf{k} \tag{5.57}$$

$$GE5: \quad \mathbf{x_{k+1}} = \frac{3}{2} \cdot \mathbf{x_k} + \frac{2}{5} \cdot \mathbf{x_{k-1}} - \frac{7}{5} \cdot \mathbf{x_{k-2}} + \frac{3}{5} \cdot \mathbf{x_{k-3}} - \frac{1}{10} \cdot \mathbf{x_{k-4}} + \frac{6}{5} \cdot h^2 \cdot \ddot{\mathbf{x}}_\mathbf{k} \tag{5.58}$$

$$GI2: \quad \mathbf{x_{k+1}} = 2 \cdot \mathbf{x_k} - \mathbf{x_{k-1}} + h^2 \cdot \ddot{\mathbf{x}}_\mathbf{k+1} \tag{5.59}$$

$$GI3: \quad \mathbf{x_{k+1}} = \frac{5}{2} \cdot \mathbf{x_k} - 2 \cdot \mathbf{x_{k-1}} + \frac{1}{2} \cdot \mathbf{x_{k-2}} + \frac{1}{2} \cdot h^2 \cdot \ddot{\mathbf{x}}_\mathbf{k+1} \tag{5.60}$$

$$GI4: \quad \mathbf{x_{k+1}} = \frac{104}{35} \cdot \mathbf{x_k} - \frac{114}{35} \cdot \mathbf{x_{k-1}} + \frac{56}{35} \cdot \mathbf{x_{k-2}} - \frac{11}{35} \cdot \mathbf{x_{k-3}} + \frac{12}{35} \cdot h^2 \cdot \ddot{\mathbf{x}}_\mathbf{k+1} \tag{5.61}$$

$$GI5: \quad \mathbf{x_{k+1}} = \frac{154}{45} \cdot \mathbf{x_k} - \frac{214}{45} \cdot \mathbf{x_{k-1}} + \frac{52}{15} \cdot \mathbf{x_{k-2}} - \frac{61}{45} \cdot \mathbf{x_{k-3}} + \frac{2}{9} \cdot \mathbf{x_{k-4}} + \frac{12}{45} \cdot h^2 \cdot \ddot{\mathbf{x}}_\mathbf{k+1} \tag{5.62}$$

The algorithms can be summarized using the following α– and β–matrices:

$$\alpha_{\mathrm{GE}} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ \frac{12}{11} \\ \frac{6}{5} \end{pmatrix} \quad ; \quad \beta_{\mathrm{GE}} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 2 & -1 & 0 & 0 & 0 \\ 2 & -1 & 0 & 0 & 0 \\ \frac{20}{11} & -\frac{6}{11} & -\frac{4}{11} & \frac{1}{11} & 0 \\ \frac{3}{2} & \frac{2}{5} & -\frac{7}{5} & \frac{3}{5} & -\frac{1}{10} \end{pmatrix} \tag{5.63}$$

$$\alpha_{\mathrm{GI}} = \begin{pmatrix} 0 \\ 1 \\ \frac{1}{2} \\ \frac{12}{35} \\ \frac{12}{45} \end{pmatrix} \quad ; \quad \beta_{\mathrm{GI}} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 2 & -1 & 0 & 0 & 0 \\ \frac{5}{2} & -2 & \frac{1}{2} & 0 & 0 \\ \frac{104}{35} & -\frac{114}{35} & \frac{56}{35} & -\frac{11}{35} & 0 \\ \frac{154}{45} & -\frac{214}{45} & \frac{52}{15} & -\frac{61}{45} & \frac{2}{9} \end{pmatrix} \tag{5.64}$$

Unfortunately, all of these methods have **F**–matrices that are even functions in $\mathbf{A} \cdot h$. Thus, none of these methods can be expected to offer an asymptotic region for eigenvalues located along the real axis. In fact, all of the above techniques are unstable everywhere in the vicinity of the origin, with the exception of the imaginary axis itself, where they exhibit marginal stability, as they should.

Let us plot the damping properties of these algorithms up and down along the imaginary axis. The damping properties of GE4 and GE5 are shown in Fig. 5.7.

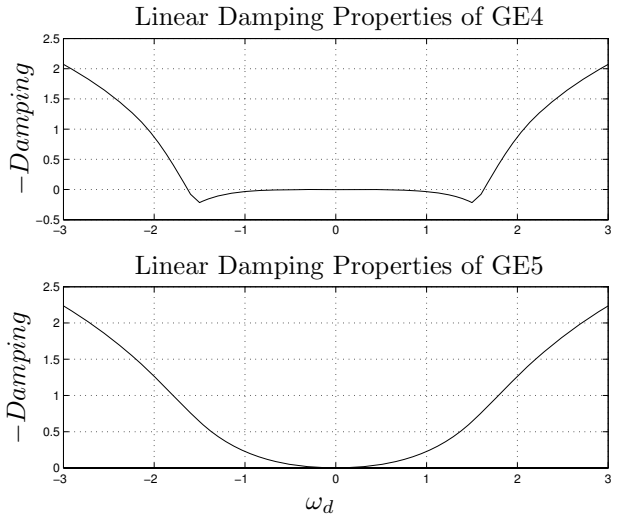

FIGURE 5.7. Damping properties of GE4 and GE5 along the imaginary axis.

Both of these algorithms may be used. Unfortunately, all of the explicit Godunov schemes, when used as stand–alone algorithms, are only applicable to the simulation of *linear conservation laws*. Engineers will likely shrug them off, because there aren't many real–life engineering applications that call for the simulation of linear conservation laws.

Yet, these algorithms may be perfectly suitable as part of blending algorithms.

Let us now discuss the implicit Godunov schemes. Their damping properties along the imaginary axis are shown in Fig. 5.8.

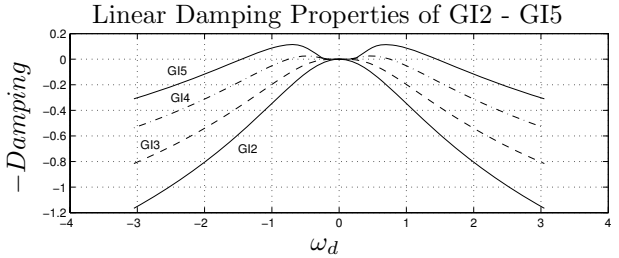

FIGURE 5.8. Damping properties of GI2 . . . GI5 along the imaginary axis.

All of these algorithms could be used as well for the simulation of linear conservation laws, but there is no good reason, why we would ever want

to do so. These algorithms have no advantages over their explicit brethren. They are only less efficient.

Furthermore, there is something else wrong with these algorithms. To understand, what that is, let us look at the damping order star of GI5.

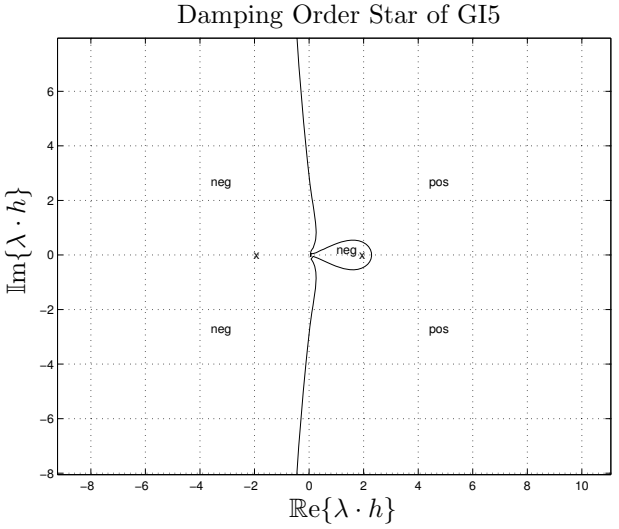

FIGURE 5.9. Damping order star of GI5 algorithm.

The locus of correct damping approximates the imaginary axis, which is nice. However, all of the algorithms of the GI class have a pole pair symmetric to the imaginary axis. Thus, all of these algorithms have a pole in the left half plane, which is something we should shun away from.

Let us now discuss the enhanced algorithms, i.e., the ones that also compute the velocity vector. We begin with GE4/AB3, and compare this algorithm with AB4.

Figure 5.10 compares the stability domains of the two algorithms.

Just as in the case of GE3/AB2, also GE4/AB3 will perform similarly to AB4 in the case of systems with their eigenvalues located close to the negative real axis, but the algorithm will outperform AB4 by leaps and bounds when employed to simulating systems with their dominant eigenvalues located near the imaginary axis.

The GE5/AB4 algorithm is unfortunately unstable. Even GE5/ABM4 turns out to be unstable.

It makes sense to try enhancing the implicit Godunov schemes by computing the velocity vector using a BDF formula of one order below that of the Godunov scheme itself.

For example, the GI3/BDF2 algorithm may be written as:

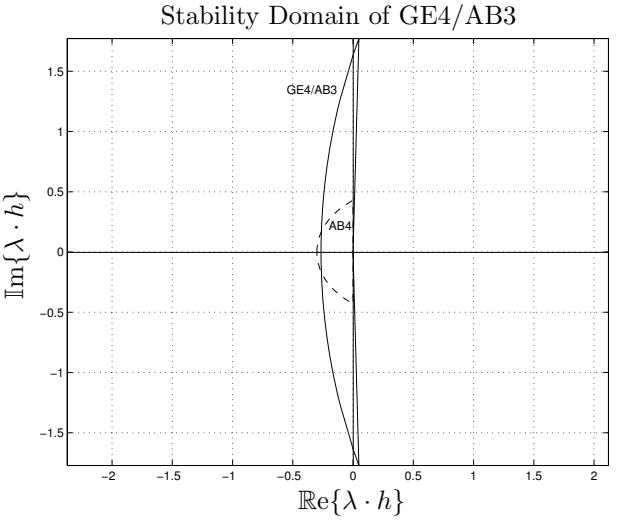

FIGURE 5.10. Stability domain of GE4/AB3.

$$\mathbf{x_{k+1}} = \frac{5}{2} \cdot \mathbf{x_k} - 2 \cdot \mathbf{x_{k-1}} + \frac{1}{2} \cdot \mathbf{x_{k-2}} + \frac{h^2}{2} \cdot \ddot{\mathbf{x}}_{\mathbf{k+1}} \tag{5.65}$$

$$h \cdot \dot{\mathbf{x}}_{\mathbf{k+1}} = \frac{4 \cdot h}{3} \cdot \dot{\mathbf{x}}_{\mathbf{k}} - \frac{h}{3} \cdot \dot{\mathbf{x}}_{\mathbf{k-1}} + \frac{2 \cdot h^2}{3} \cdot \ddot{\mathbf{x}}_{\mathbf{k+1}} \tag{5.66}$$

Unfortunately, it didn't work. Some of the resulting algorithms are indeed A–stable. They have nice unstable regions in the right half complex $\lambda \cdot h$ plane, but unfortunately, the damping is exactly equal to zero everywhere else. None of these algorithms produces an asymptotic region everywhere around the origin.

5.8 The Newmark Algorithm

An integration algorithm for the direct numerical solution of second derivative systems that has seen quite a bit of publicity in the mechanical engineering literature is the integration algorithm by Newmark [5.7]. Although the algorithm had originally been proposed for the solution of problems in structural dynamics, such as the simulation of earthquakes, it can be used for the simulation of other mechanical systems as well [5.1, 5.5].

The method can be described as follows:

$$\mathbf{x_{k+1}} = \mathbf{x_k} + h \cdot \dot{\mathbf{x}}_{\mathbf{k}} + \frac{h^2}{2} \cdot [(1 - \vartheta_1) \cdot \ddot{\mathbf{x}}_{\mathbf{k}} + \vartheta_1 \cdot \ddot{\mathbf{x}}_{\mathbf{k+1}}] \tag{5.67}$$

$$h \cdot \dot{\mathbf{x}}_{\mathbf{k+1}} = h \cdot \dot{\mathbf{x}}_{\mathbf{k}} + h^2 \cdot [(1 - \vartheta_2) \cdot \ddot{\mathbf{x}}_{\mathbf{k}} + \vartheta_2 \cdot \ddot{\mathbf{x}}_{\mathbf{k+1}}] \tag{5.68}$$

The method is clearly second–order accurate, as the solution for $\mathbf{x_{k+1}}$ approximates the Taylor–Series directly up to the quadratic term, whereas the solution for $\dot{\mathbf{x}}_{\mathbf{k+1}}$ approximates the Taylor–Series up to the linear term. Since the velocity vector gets always multiplied by the step size, h, the overall method must be second–order accurate.

It is a ϑ–method with two fudge parameters, ϑ_1 and ϑ_2. For $\vartheta_1 = \vartheta_2 = 0$, the method is explicit; for all other combinations of ϑ_1 and ϑ_2, the method is implicit.

The Newmark algorithm is different from the implicit Godunov techniques. The Godunov algorithms made it a point, not to make use of the velocity vector in the computation of the position vector. This makes sense in the special case of velocity–free second–derivative systems, but doesn't make much sense otherwise, i.e., the enhanced Godunov methods, that also compute the velocity vector, added an unnecessary constraint on the design of the algorithm that we paid for bitterly, since we were fighting even functions that prevented us from getting asymptotic regions around the origin.

Let us plot the stability domains of the Newmark algorithm for $\vartheta_1 = \vartheta_2 = \{0.0, 0.25, 0.5, 0.75, 1.0\}$.

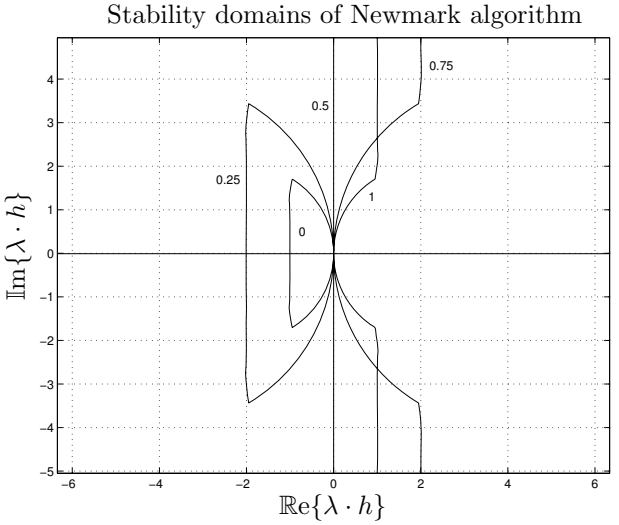

FIGURE 5.11. Stability domains of Newmark algorithm as a function of ϑ_1 and ϑ_2.

These are interesting looking and quite unusual stability domains. The algorithms are symmetric to $\vartheta_1 = \vartheta_2 = 0.5$. For $\vartheta_1 = \vartheta_2 = 0$, the algorithm exhibits a stable region in the left half plane that looks like an ascending half moon. The stable region is limited by $\mathbb{R}\mathrm{e}\{\lambda \cdot h\} \geq -1.0$. As ϑ_1 and ϑ_2 increase their values, the half moon grows in size, and the stable region

extends further to the left. For $\vartheta_1 = \vartheta_2 = 0.25$, the region is limited by $\mathbb{R}\mathrm{e}\{\lambda \cdot h\} \geq -2.0$. As ϑ_1 and ϑ_2 approach values of 0.5, the entire left half plane is covered by the stable region. For $\vartheta_1 = \vartheta_2 = 0.5$, the resulting algorithm is F–stable. At $\vartheta_1 = \vartheta_2 = 0.5$, the stable region wraps around infinity. For $\vartheta_1 = \vartheta_2 = 0.75$, the entire left half plane is stable, but also the region limited by $\mathbb{R}\mathrm{e}\{\lambda \cdot h\} \geq 2.0$.

¿From a practical perspective, the algorithms with $\vartheta_1 = \vartheta_2 \in (0, 0.5)$ are probably not of much interest. $\vartheta_1 = \vartheta_2 = 0$ could be interesting, as this is an explicit algorithm. Also the algorithms with $\vartheta_1 = \vartheta_2 \geq 0.5$ are of interest, since they are all A–stable.

Let us look at some damping plots. Figure 5.12 shows the linear and logarithmic damping plots of the F–stable Newmark algorithm for $\vartheta_1 = \vartheta_2 = 0.5$.

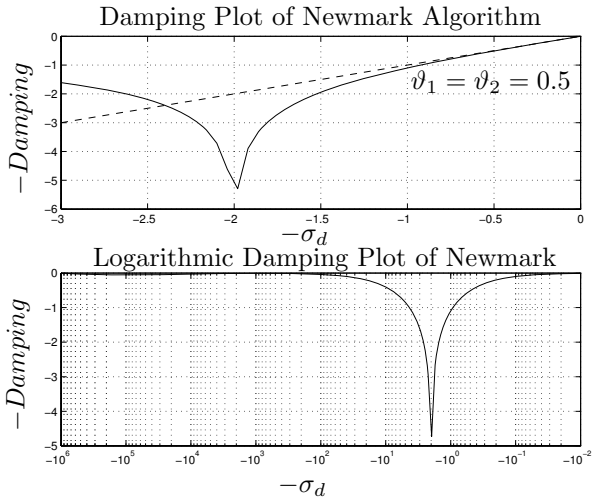

FIGURE 5.12. Damping plot of Newmark algorithm with $\vartheta_1 = \vartheta_2 = 0.5$.

As was to be expected, the damping is zero everywhere at infinity, and in fact, the damping assumes very small values in large portions of the left half plane, which could potentially become a problem when dealing with stiff systems.

Let us check, whether the Newmark algorithms with larger ϑ–values fare any better. Figure 5.13 shows the damping plots of the A–stable Newmark algorithm for $\vartheta_1 = \vartheta_2 = 0.75$.

It did not help. The damping still approaches zero at infinity. We should have been able to predict this result from the stability domain alone. Since the border of stability reaches all the way to infinity at some place, the numerical scheme must exhibit marginal stability at infinity, irrespective of how infinity is being approached.

Furthermore, the algorithm with $\vartheta_1 = \vartheta_2 = 0.75$ exhibits a much smaller

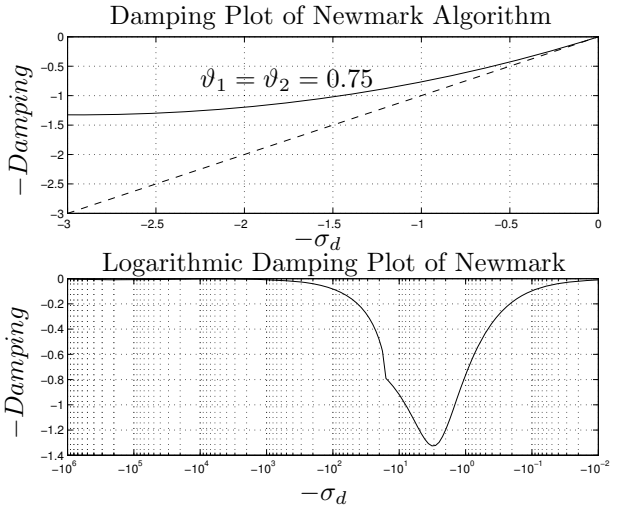

FIGURE 5.13. Damping plot of Newmark algorithm with $\vartheta_1 = \vartheta_2 = 0.75$.

asymptotic region. Whereas Fig.5.12 showed an asymptotic region of approximately 0.7, Fig.5.13 shows an asymptotic region of approximately 0.2. This result is disappointing.

We shall refrain from plotting more damping plots. The damping plots for $\vartheta_1 = \vartheta_2 = 1.0$ don't look any more promising. Hence the Newmark family of algorithms is better suited for the simulation of systems exhibiting oscillatory behavior, such as earthquakes or elastic systems, than for the simulation of stiff problems.

5.9 Summary

In this chapter, we have looked at new classes of simulation algorithms that can deal with second derivative systems directly, i.e., without first converting them to state–space form.

Our original goal had been to design algorithms for velocity–free problems that would avoid the seemingly unnecessary computation of the velocity vector altogether. To this end, we developed two families of algorithms, the explicit and implicit Godunov schemes, using our old friends, the Newton–Gregory polynomials.

Unfortunately, this approach was unsuccessful. None of the resulting algorithms possess any asymptotic regions around their origins, as the **F**–matrices associated with these algorithms are even functions of $\mathbf{A} \cdot h$. Consequently, these algorithms, when used in a stand–alone mode, can only be employed for the simulation of linear conservation laws, a result that is not overly exciting.

We then enhanced these algorithms to make them suitable for the simulation of systems with damping, and found two explicit algorithms, GE3/AB2 and GE4/AB3, that can be considered serious contenders of AB3 and AB4 for the simulation of mechanical systems with strong oscillatory behavior. This is certainly a nice, albeit still not a truly exciting, result.

We then analyzed an algorithm that has seen quite a bit of publicity in the mechanical engineering literature: the algorithm by Newmark, an algorithm that is still being used quite frequently for the simulation of mechanical systems exhibiting oscillatory behavior. Unfortunately, the algorithm is only second–order accurate, which makes it unsuitable for general purpose simulation, as most engineering applications call for third and fourth–order accurate integration algorithms, in order to be simulated efficiently.

The Newmark family of algorithms can be, and have often been, used in *real–time applications*, because most real–time applications call for small step sizes to track external input signals, and therefore, are often simulated using low–order integration algorithms. Since real–time applications cannot deal with implicit algorithms very well, we should probably use the linearly–implicit variant of the Newmark algorithm in those cases. We shall deal intensively and extensively with real–time simulation in Chapter 10 of this book.

... And here comes the most exciting aspect of this chapter. Nathan Newmark, in 1959, accomplished for second derivative systems, what Leonhard Euler accomplished for first derivative systems (i.e., state–space models) in 1768 [5.4], and yet, researchers still write articles about this algorithm today, and the algorithm is still being employed frequently in engineering applications.

In the previous two chapters, we had to work very hard to make a mark. The field of numerical ODE solvers for state–space models is a very mature research topic, and consequently, it is difficult to come up with something new that no–one else has thought about before.

In contrast, the field of numerical ODE solvers for second derivative systems is still in its infancy, largely ignored by the community of applied mathematicians so far, and is therefore a fruitful research area for young and aspiring researchers of numerical methods.

5.10 References

[5.1] Klaus-Jürgen Bathe. *Finite Element Procedures in Engineering Analysis.* Prentice–Hall, 1982.

[5.2] Christopher Paul Beamis. Solution of Second Order Differential Equations Using the Godunov Integration Method. Master's thesis, Dept. of Electrical & Computer Engineering, University of Arizona, Tucson, Ariz., 1990.

[5.3] François E. Cellier. *Continuous System Modeling*. Springer Verlag, New York, 1991. 755p.

[5.4] Leonhard Euler. De integratione æquationum differentialium per approximationem. In *Opera Omnia*, volume 11 of *first series*, pages 424–434. Institutiones Calculi Integralis, Teubner Verlag, Leipzig, Germany, 1913.

[5.5] Javier Garcia de Jalón and Eduardo Bayo. *Kinematic and Dynamic Simulation of Multibody Systems –The Real–Time Challenge–*. Wiley, 1994.

[5.6] John C. Houbolt. A Recurrence Matrix Solution for the Dynamic Response of Elastic Aircraft. *Journal of Aeronautical Science*, 17:540–550, 1950.

[5.7] Nathan M. Newmark. A Method of Computation for Structural Dynamics. *ASCE Journal of the Engineering Mechanics Division*, pages 67–94, 1959.

[5.8] Raymond T. Stefani, Clement J. Savant Jr., Bahram Shahian, and Gene H. Hostetter. *Design of Feedback Control Systems*. Saunders College Publishing, Orlando, Florida, 1994. 819p.

[5.9] Edward L. Wilson. A Computer Program for the Dynamic Stress Analysis of Underground Structures. Technical Report SESM Report, 68-1, University of California, Berkeley, Division of Structural Engineering and Structural Mechanics, 1968.

5.11 Bibliography

[B5.1] Edda Eich-Söllner and Claus Führer. *Numerical Methods in Multibody Dynamics*. Teubner–Verlag, Stuttgart, Germany, 1998.

[B5.2] Michel Géradin and Alberto Gardona. *Flexible Multibody Dynamics: A Finite Element Approach*. John Wiley & Sons, Chichester, New York, 2001.

[B5.3] Parviz E. Nikravesh. *Computer–aided Analysis of Mechanical Systems*. Prentice Hall, Englewood Cliffs, New Jersey, 1988. 370p.

5.12 Homework Problems

[H5.1] Explicit Godunov Algorithms

We wish to generate the class of GE algorithms introduced in this chapter. These linear multi–step algorithms compute $\mathbf{x_{k+1}}$ as a function of previous

values of $\mathbf{x}$ and of $\ddot{\mathbf{x}}_\mathbf{k}$.

Develop $\mathbf{x}(t)$ into a Newton–Gregory backward polynomial around the time instant t_{k+1}. Compute the second derivative:

$$\ddot{\mathbf{x}}(t) = \frac{1}{h^2} \cdot \mathbf{f}(s) \tag{H5.1a}$$

Evaluate this expression for $s = -1$. Truncating this expression after the quadratic ∇^2 term, you obtain a second–order accurate formula for $\ddot{\mathbf{x}}_\mathbf{k}$, which can be solved for $\mathbf{x}_{\mathbf{k+1}}$. Truncating after the cubic ∇^3 term, you obtain a third–order accurate formula, etc.

There is no first–order accurate GE scheme. From (H5.1a), one can recognize by inspection that the second–order accurate GE2 scheme is already third–order accurate. Explain.

[H5.2] Implicit Godunov Algorithms

We wish to generate the class of GI algorithms introduced in this chapter. These linear multi–step algorithms compute $\mathbf{x}_{\mathbf{k+1}}$ as a function of previous values of $\mathbf{x}$ and of $\ddot{\mathbf{x}}_{\mathbf{k+1}}$.

Develop $\mathbf{x}(t)$ into a Newton–Gregory backward polynomial around the time instant t_{k+1}. Compute the second derivative:

$$\ddot{\mathbf{x}}(t) = \frac{1}{h^2} \cdot \mathbf{f}(s) \tag{H5.2a}$$

Evaluate this expression for $s = 0$. Truncating this expression after the quadratic ∇^2 term, you obtain a second–order accurate formula for $\ddot{\mathbf{x}}_{\mathbf{k+1}}$, which can be solved for $\mathbf{x}_{\mathbf{k+1}}$. Truncating after the cubic ∇^3 term, you obtain a third–order accurate formula, etc.

[H5.3] Stability Domain of GE4/AB3

The method introduced in earlier chapters for drawing stability domains was geared towards linear time–invariant homogeneous multi–variable state–space models:

$$\dot{\mathbf{x}} = \mathbf{A} \cdot \mathbf{x} \tag{H5.3a}$$

We generated real–valued $\mathbf{A}$–matrices $\in \mathbb{R}^{2\times 2}$ with their eigenvalues located on the unit circle, at an angle α away from the negative real axis. We then computed the $\mathbf{F}$–matrix corresponding to that $\mathbf{A}$–matrix for the given algorithm, and found the largest value of the step size h, for which all eigenvalues of $\mathbf{F}$ remained inside the unit circle. This gave us one point on the stability domain. We repeated this procedure for all suitable values of the angle α.

The algorithm needs to be modified for dealing with second derivative systems described by the linear time–invariant homogeneous multi–variable second–derivative model:

$$\ddot{\mathbf{x}} = \mathbf{A}^2 \cdot \mathbf{x} + \mathbf{B} \cdot \dot{\mathbf{x}} \tag{H5.3b}$$

We need to find real–valued **A**– and **B**–matrices such that the model of (H5.3b) has its eigenvalues located on the unit circle.

This can be accomplished using the scalar model:

$$\ddot{x} = a^2 \cdot x + b \cdot \dot{x} \tag{H5.3c}$$

where:

$$a = \sqrt{a_{21}} \tag{H5.3d}$$
$$b = a_{22} \tag{H5.3e}$$

of the formerly used **A**–matrix.

Write the GE4/AB3 algorithm as follows:

$$\begin{aligned} x_{k+1} &= \frac{20}{11} \cdot x_k - \frac{6}{11} \cdot x_{k-1} - \frac{4}{11} \cdot x_{k-2} + \frac{1}{11} \cdot x_{k-3} \\ &\quad + \frac{12 \cdot h^2}{11} \cdot \ddot{x}_k \qquad \text{(H5.3f)} \\ h \cdot \dot{x}_{k+1} &= h \cdot \dot{x}_k + \frac{23 \cdot h^2}{12} \cdot \ddot{x}_k - \frac{4 \cdot h^2}{3} \cdot \ddot{x}_{k-1} \\ &\quad + \frac{5 \cdot h^2}{12} \cdot \ddot{x}_{k-2} \qquad \text{(H5.3g)} \\ \ddot{x} &= a^2 \cdot x + b \cdot \dot{x} \qquad \text{(H5.3h)} \end{aligned}$$

Substitute Eq.(H5.3h) into Eq.(H5.3f) and Eq.(H5.3g), and rewrite the resulting equations in a state–space form:

$$\xi_{\mathbf{k+1}} = \mathbf{F} \cdot \xi_{\mathbf{k}} \tag{H5.3i}$$

whereby the state vector ξ is chosen as:

$$\xi_{\mathbf{k}} = \begin{pmatrix} x_{k-3} \\ h \cdot \dot{x}_{k-3} \\ x_{k-2} \\ h \cdot \dot{x}_{k-2} \\ x_{k-1} \\ h \cdot \dot{x}_{k-1} \\ x_k \\ h \cdot \dot{x}_k \end{pmatrix} \tag{H5.3j}$$

The **F**–matrix turns out to be a function of $(a \cdot h)^2$ and of $b \cdot h$.

The remainder of the algorithm remains the same as before. Draw the stability domain of GE4/AB3 using this approach.

[H5.4] Stability Domain of GI3/BDF2

We want to draw the stability domain of GI3/BDF2. However, we shall use a different approach from that advocated in Hw.[H5.3], an approach that is a bit slower in execution, but more general.

We start by writing the algorithm as follows:

$$x_{k+1} = \frac{5}{2} \cdot x_k - 2 \cdot x_{k-1} + \frac{1}{2} \cdot x_{k-2} + \frac{h^2}{2} \cdot \ddot{x}_{k+1} \quad \text{(H5.4a)}$$

$$h \cdot \dot{x}_{k+1} = \frac{4 \cdot h}{3} \cdot \dot{x}_k - \frac{h}{3} \cdot \dot{x}_{k-1} + \frac{2 \cdot h^2}{3} \cdot \ddot{x}_{k+1} \quad \text{(H5.4b)}$$

$$\ddot{x} = a^2 \cdot x + b \cdot \dot{x} \quad \text{(H5.4c)}$$

We substitute Eq.(H5.4c) into Eq.(H5.4a) and Eq.(H5.4b), and rewrite the resulting equations in a state–space form:

$$\xi_{\mathbf{k+1}} = \mathbf{F} \cdot \xi_{\mathbf{k}} \quad \text{(H5.4d)}$$

whereby the state vector ξ is chosen as:

$$\xi_{\mathbf{k}} = \begin{pmatrix} x_{k-2} \\ h \cdot \dot{x}_{k-2} \\ x_{k-1} \\ h \cdot \dot{x}_{k-1} \\ x_k \\ h \cdot \dot{x}_k \end{pmatrix} \quad \text{(H5.4e)}$$

The **F**–matrix turns out to be a function of $(a \cdot h)^2$ and of $b \cdot h$.

We choose a and b such that the two eigenvalues of the second–order differential equation are located in the position:

$$\lambda_{1,2} = \sigma \pm j \cdot \omega \quad \text{(H5.4f)}$$

We perform a double loop over σ and ω to cover an entire area of the complex plane with eigenvalue locations.

For each eigenvalue location, we compute the corresponding values of α and h, compute the corresponding **F**–matrix, and determine the discrete damping value using the equation:

$$damp = -\log(\max(\text{abs}(\text{eig}(\mathbf{F})))) \quad \text{(H5.4g)}$$

The damping value can be interpreted as a real-valued function of the complex–valued argument $\sigma + j \cdot \omega$.

Use MATLAB's contour plot to draw the locus of all points of zero damping. This will be the stability domain of the method.

Interpret the results that you get.

[H5.5] Cervical Syndrome

Given the model of a sitting human body presented in Fig. 5.1. We exert this model by a sinusoidal force of 1.5 Hz. Simulate this model during 10 seconds, once using the GE4/AB3 algorithm, and once using the AB4 algorithm. Choose a fixed step size of $h = 0.1$ seconds.

Determine, which of the two algorithms is faster, i.e., requires a smaller number of floating point operations.

Assume that MATLAB's solution using the ODE45 algorithm of variable step size with a relative tolerance of 0.0001 is accurate, and compare the two solutions of your own simulation algorithms against MATLAB's solution. Determine, which of the algorithms produced more accurate results.

5.13 Projects

[P5.1] Houbolt's Integration Algorithm

John Houbolt proposed already in 1950 a second–derivative integration algorithm [5.6] that is very similar to the GI3/BDF2 method introduced in this chapter. Houbolt's algorithm can be written as follows:

$$\mathbf{x_{k+1}} = \frac{5}{2} \cdot \mathbf{x_k} - 2 \cdot \mathbf{x_{k-1}} + \frac{1}{2} \cdot \mathbf{x_{k-2}} + \frac{h^2}{2} \cdot \ddot{\mathbf{x}}_{\mathbf{k+1}} \quad \text{(P5.1a)}$$

$$h \cdot \dot{\mathbf{x}}_{\mathbf{k+1}} = \frac{11}{6} \cdot \mathbf{x_{k+1}} - 3 \cdot \mathbf{x_k} + \frac{3}{2} \cdot \mathbf{x_{k-1}} - \frac{1}{3} \cdot \mathbf{x_{k-2}} \quad \text{(P5.1b)}$$

The second derivative formula of Houbolt's algorithm can immediately be identified as GI3. The formula used for the velocity vector is BDF3; however, the formula was used differently from the way, it had been employed by us in the description of the GI3/BDF2 algorithm. Clearly, the Houbolt algorithm is third–order accurate.

Although it would have sufficed to use BDF2 for the velocity vector, nothing would have been gained computationally by choosing the reduced–order algorithm.

We can transform the Houbolt algorithm to the form that we meanwhile got used to by substituting Eq.(P5.1a) into Eq.(P5.1b). The so rewritten Houbolt algorithm assumes the form:

$$\mathbf{x_{k+1}} = \frac{5}{2} \cdot \mathbf{x_k} - 2 \cdot \mathbf{x_{k-1}} + \frac{1}{2} \cdot \mathbf{x_{k-2}} + \frac{h^2}{2} \cdot \ddot{\mathbf{x}}_{\mathbf{k+1}} \quad \text{(P5.1c)}$$

$$h \cdot \dot{\mathbf{x}}_{\mathbf{k+1}} = \frac{19}{12} \cdot \mathbf{x_k} - \frac{13}{6} \cdot \mathbf{x_{k-1}} + \frac{7}{12} \cdot \mathbf{x_{k-2}} + \frac{11 \cdot h^2}{12} \cdot \ddot{\mathbf{x}}_{\mathbf{k+1}} \quad \text{(P5.1d)}$$

Find the stability domain and damping plot of Houbolt's algorithm, and discuss the properties of this algorithm in the same way, as Nemark's algorithm was discussed in this chapter.

Repeat the analysis, replacing the BDF3 formula by the BDF2 formula.

Analyze whether stable GI4/BDF3 and/or GI4/BDF4 algorithms can be constructed in the same fashion.

[P5.2] Wilson's Integration Algorithm

In 1968, Wilson proposed yet another second–derivative integration algorithm [5.9] that is quite similar to Newmark's method. A clean derivation of Wilson's algorithm can be found in Bathe [5.1].

The algorithm can be converted easily to the form that we embraced in this chapter:

$$\mathbf{x_{k+1}} = \mathbf{x_k} + h \cdot \dot{\mathbf{x}}_\mathbf{k} + \frac{h^2}{6} \cdot \ddot{\mathbf{x}}_\mathbf{k+1} + \frac{h^2}{3} \cdot \ddot{\mathbf{x}}_\mathbf{k} \qquad \text{(P5.2a)}$$

$$h \cdot \dot{\mathbf{x}}_\mathbf{k+1} = h \cdot \dot{\mathbf{x}}_\mathbf{k} + \frac{h^2}{2} \cdot \ddot{\mathbf{x}}_\mathbf{k+1} + \frac{h^2}{2} \cdot \ddot{\mathbf{x}}_\mathbf{k} \qquad \text{(P5.2b)}$$

Clearly, Eq.(P5.2a) is second–order accurate in $\mathbf{x_k}$, whereas Eq.(P5.2b) is first–order accurate in $\dot{\mathbf{x}}_\mathbf{k}$. Hence the overall algorithm is second–order accurate.

Find the stability domain and damping plot of Wilson's algorithm, and discuss the properties of this algorithm in the same way, as Nemark's algorithm was discussed in this chapter.

Compare Wilson's and Newmark's algorithms with each other. Which of them would you use when and why?

5.14 Research

[R5.1] Second–derivative Runge–Kutta Algorithms

When designing explicit Runge–Kutta algorithms, we started out with a partial step of Euler, used the result obtained by that stage as a predictor, and added another stage as a corrector, in which we blended the solutions of the two stages. We continued in the same way to more and more stages, trying to approximate the Taylor–series expansion to increasingly higher orders. Contrary to the multi–step methods, we were able to determine also the nonlinear order of approximation accuracy in the case of the Runge–Kutta algorithms.

It must be possible to generalize the Runge–Kutta algorithms to second–derivative form. To this end, we postulate a first partial step using the algorithm:

$$\mathbf{x}^{\mathrm{P}_1}(t + \alpha_1 \cdot h) = \mathbf{x_k} + \alpha_1 \cdot h \cdot \dot{\mathbf{x}}_\mathbf{k} + \frac{(\alpha_1 \cdot h)^2}{2} \cdot \ddot{\mathbf{x}}_\mathbf{k} \tag{R5.1a}$$

We then proceed to building corrector stages in the same fashion, as we did in the case of the regular RK algorithms.

Whereas we left the first derivative alone in the case of the RK algorithms, and only expanded second and higher–order derivatives into Taylor series, we now must leave the first and second derivatives alone, and only develop third and higher–order derivatives into Taylor series.

We shall use a regular RK algorithm of one order lower than the second–derivative algorithm to compute the velocity vector. We use the freedom in the design of such algorithms to ensure that the individual stages of the second–derivative algorithm for computing the position vector and the first–derivative algorithm for computing the velocity vector are evaluated at the same instants of simulated time, i.e.:

$$\alpha_i(2^{\mathrm{nd}} \text{ derivative algorithm}) = \alpha_i(1^{\mathrm{st}} \text{ derivative algorithm}) \tag{R5.1b}$$

Clearly, there don't exist first–order accurate generalized RK algorithms, and the second–order accurate generalized explicit RK algorithm, GERK2, can be written as follows:

$$\mathbf{x_{k+1}} = \mathbf{x_k} + h \cdot \dot{\mathbf{x}}_\mathbf{k} + \frac{h^2}{2} \cdot \ddot{\mathbf{x}}_\mathbf{k} \tag{R5.1c}$$

$$h \cdot \dot{\mathbf{x}}_\mathbf{k+1} = h \cdot \dot{\mathbf{x}}_\mathbf{k} + h^2 \cdot \ddot{\mathbf{x}}_\mathbf{k} \tag{R5.1d}$$

which is identical to the Newmark algorithm with $\vartheta_1 = \vartheta 2 = 0.0$.

It should be possible to develop second–derivative (generalized) explicit Runge–Kutta algorithms of any order, and we would expect that these algorithms should outperform their regular RK cousins when dealing with nonlinear mechanical systems that exhibit highly oscillatory behavior.