CHAPTER 10: *Numerical Methods for DAEs*

Numerical approaches for the solution of DAEs divide roughly into two classes:

1. direct discretization

2. reformulation (index reduction) plus discretization

Direct discretization is desirable because reformulation may be costly and require additional user input and/or intervention.

However, direct discretizations are only feasible for index-1 and semi-explicit index-2 DAEs!

This makes reformulation popular in practice.

Fortunately, many DAEs in practice are index-1 or simple combinations of Hessenberg systems.

We consider two classes of problems:

1. fully implicit index-1 DAEs:

$$\mathbf{F}(t, \mathbf{y}, \dot{\mathbf{y}}) = \mathbf{0}.$$

2. Hessenberg (pure) index-2 form

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{f}(t, \mathbf{x}, \mathbf{z}), \\ \mathbf{0} &= \mathbf{g}(t, \mathbf{x}). \end{aligned}$$

Recall, the class of semi-explicit index-2 DAEs

$$\dot{\mathbf{x}} = \mathbf{f}(t, \mathbf{x}, \mathbf{z}), \tag{1a}$$

$$\mathbf{0} = \mathbf{g}(t, \mathbf{x}, \mathbf{z}), \tag{1b}$$

is equivalent to the class of fully implicit index-1 DAEs.

Of course, the conversion may come at a price of increased system size.

For the index-2 DAE, the algebraic variables $\mathbf{z}$ may be index-1 or index-2.

For the Hessenberg form, all the $\mathbf{z}$ are index-2.

(That is why we call it the *pure* index-2 form.)

There are some convergence results for numerical methods applied to even higher-index Hessenberg DAEs, but there are difficulties in constructing general-purpose codes.

The best strategy seems to be to perform index reduction and solve the resulting lower-index problem with a suitable discretization. (More about this later.)

10.1 Direct Discretizations

Consider the *regularization* of the semi-explicit index-2 DAE by replacing the algebraic constraint by the ODE

$$\epsilon \dot{\mathbf{z}} = \mathbf{g}(t, \mathbf{x}, \mathbf{z}),$$

where $0 \leq \epsilon \ll 1$.

Note: We do not intend to carry out this process!

The very stiff ODE so obtained is usually more trouble to solve than the original DAE!

But this allows us to think about what properties are required of a numerical method to be good for DAEs.

- Because the regularized ODE is stiff, we consider numerical methods for stiff ODEs.

- Methods that possess the property of stiff decay are particularly attractive.

Recall these methods tend to skip over transient regions. This is exactly what we would like to do to: skip over the "artificial" layers induced by the regularization and transition to the limit $\epsilon \to 0$.

All the good methods for DAEs will have stiff decay, but this property alone is not sufficient for success.

For initial-value DAEs that are hard to transform or have a stiff underlying ODE, BDF and Radau methods are preferred.

So we begin with the grandfather of both of these families: backward Euler.

10.1.1 Backward Euler

Consider the general DAE

$$\mathbf{F}(t, \mathbf{y}, \dot{\mathbf{y}}) = \mathbf{0}.$$

Apply backward Euler directly to this DAE:

$$\mathbf{F}\left(t_n, \mathbf{y}_n, \frac{\mathbf{y}_n - \mathbf{y}_{n-1}}{\Delta t_n}\right) = \mathbf{0}.$$

$\rightarrow$ a system of m nonlinear equations for $\mathbf{y}_n$ at each time t_n.

Sadly, *this simple method does not always work!*

Worst case: There are simple high-index DAEs with well-defined solutions for which backward Euler (and in fact all other multi-step and Runge–Kutta methods!) are unstable or not defined.

Example 10.1

Consider the following linear index-2 DAE:

$$\begin{bmatrix} 0 & 0 \\ 1 & \eta t \end{bmatrix} \dot{\mathbf{y}} + \begin{bmatrix} 1 & \eta t \\ 0 & 1+\eta \end{bmatrix} \mathbf{y} = \begin{bmatrix} q(t) \\ 0 \end{bmatrix},$$

where η is a parameter and $q(t)$ is differentiable.

Exact solution: $y_1(t) = q(t) + \eta\, t\, \dot{q}(t)$, $y_2 = -\dot{q}(t)$.

$\rightarrow$ well defined for all η.

The problem is stable for moderate values of η, but backward Euler is not defined if $\eta = -1$! (verify!)

In fact, it can be shown that backward Euler is unstable when $\eta < -0.5$.

Some analysis of this problem offers some insight.

We can transform to semi-explicit form via

$$\mathbf{y} = \begin{bmatrix} 1 & -\eta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}.$$

This yields

$$\dot{u} + v = 0, \qquad u = q(t).$$

Note: Forward Euler is undefined for this problem!

Applying backward Euler to this yields

$$u_n = q(t_n), \qquad v_n = -\frac{q(t_n) - u_{n-1}}{\Delta t_n}.$$

Starting with a consistent initial value $u(0) = q(0)$,

$$v_n = -\dot{q}(t_n) + \mathcal{O}(\Delta t),$$

which is all you can expect from a first-order method like backward Euler.

Now apply backward Euler directly to the original DAE:

$$\begin{bmatrix} 0 & 0 \\ 1 & \eta t_n \end{bmatrix} \frac{\mathbf{y}_n - \mathbf{y}_{n-1}}{\Delta t} + \begin{bmatrix} 1 & \eta t_n \\ 0 & 1+\eta \end{bmatrix} \mathbf{y}_n = \begin{bmatrix} q(t_n) \\ 0 \end{bmatrix}.$$

Defining

$$\begin{bmatrix} u_n \\ v_n \end{bmatrix} = \begin{bmatrix} 1 & \eta t_n \\ 0 & 1 \end{bmatrix} \mathbf{y}_n,$$

we get

$$u_n = q(t_n), \qquad (1+\eta) v_n = -\frac{q(t_n) - q(t_{n-1})}{\Delta t_n}.$$

So u_n is reproduced exactly (as before), but v_n is undefined when $\eta = -1$.

Note that the transformations we have introduced have *decoupled* $\mathbf{y}$ into differential and algebraic components.

Backward Euler works well for the decoupled problem.

However, *direct discretizations of nondecoupled DAEs*

of index higher than 1 are $\mathbf{not}$ *recommended!*

For the rest of this subsection, we consider only index-1 or semi-explicit index-2 DAEs.

Starting with the semi-explicit index-1 DAE,

$$\dot{\mathbf{x}} = \mathbf{f}(t, \mathbf{x}, \mathbf{z}),$$
$$\mathbf{0} = \mathbf{g}(t, \mathbf{x}, \mathbf{z}),$$

where $\mathbf{g}_\mathbf{z}$ is nonsingular, it is easy to see that backward Euler retains all its nice properties (order, stability, convergence) from the ODE case.

From the implicit function theorem, there exists a function $\tilde{\mathbf{g}}$ such that

$$\mathbf{z} = \tilde{\mathbf{g}}(t, \mathbf{x}).$$

Assuming there is only one such $\tilde{\mathbf{g}}$, the DAE is equivalent to the ODE

$$\dot{\mathbf{x}} = \mathbf{f}(t, \mathbf{x}, \tilde{\mathbf{g}}(t, \mathbf{x})).$$

Applying backward Euler directly to the DAE, we obtain

$$\begin{aligned} \frac{\mathbf{x}_n - \mathbf{x}_{n-1}}{\Delta t_n} &= \mathbf{f}(t_n, \mathbf{x}_n, \mathbf{z}_n), \\ \mathbf{0} &= \mathbf{g}(t_n, \mathbf{x}_n, \mathbf{z}_n). \end{aligned}$$

Solving for $\mathbf{z}_n$ and substituting into the differential equation yields

$$\frac{\mathbf{x}_n - \mathbf{x}_{n-1}}{\Delta t_n} = \mathbf{f}(t_n, \mathbf{x}_n, \tilde{\mathbf{g}}(t_n, \mathbf{x}_n)).$$

This is backward Euler applied to the underlying ODE.

So all the results from the analysis of backward Euler apply now:

Backward Euler is first-order accurate, stable, and convergent for semi-explicit index-1 DAEs.

The same results can be shown for fully implicit index-1 DAEs and semi-explicit index-2 DAEs (but the analysis is quite a bit more involved).

See the text for some hints as to how this is performed.

10.1.2 BDF and Multistep Methods

In the usual notation, applying a BDF method with constant stepsize Δt to a general nonlinear DAE yields

$$\mathbf{F}\left(t_n, \mathbf{y}_n, \frac{1}{\beta_0 \Delta t}\sum_{j=0}^{k} \alpha_j \mathbf{y}_{n-j}\right) = \mathbf{0}.$$

Most software packages based on BDF methods solve fully implicit index-1 DAEs.

Fortunately, many practical problems arise in this form!

Convergence results extend naturally from that of backward Euler: a k-step BDF method with fixed Δt and $k < 7$ is convergent of order k if all initial values are accurate to $\mathcal{O}((\Delta t)^k)$ and the Newton iteration converges to $\mathcal{O}((\Delta t)^{k+1})$.

This result extends to variable-stepsize BDF methods provided the implementation is such that the method is stable for ODEs.

We also have similar convergence results for semi-explicit index-2 DAEs.

For other linear multistep methods applied to general index-1 DAEs and Hessenberg index-2 DAEs, the coefficients of the LMM must satisfy additional DAE order conditions to attain order greater than 2.

(It turns out these conditions are automatically satisfied by BDF methods.)

10.1.3 Radau Collocation and Implicit Runge–Kutta Methods

The s-stage implicit RK method applied to the general nonlinear DAE takes the form:

$$
\begin{aligned}
\mathbf{0} &= \mathbf{F}(t_{ni}, \mathbf{Y}_i, \mathbf{K}_i), \\
t_{ni} &= t_{n-1} + \Delta t_n \, c_i, \\
\mathbf{Y}_i &= \mathbf{y}_{n-1} + \Delta t_n \sum_{j=1}^{s} a_{ij} \mathbf{K}_j, \quad i = 1, 2, \ldots, s, \\
\mathbf{y}_n &= \mathbf{y}_{n-1} + \Delta t_n \sum_{i=1}^{s} b_i \mathbf{K}_i.
\end{aligned}
$$

We assume $\mathbf{A}$ is non-singular.

For the semi-explicit problem, the internal stages satisfy

$$\begin{aligned}
\mathbf{K}_i &= \mathbf{f}(t_{ni}, , \mathbf{X}_i, \mathbf{Z}_i), \\
\mathbf{X}_i &= \mathbf{x}_{n-1} + \Delta t_n \sum_{j=1}^{s} a_{ij} \mathbf{K}_j, \\
\mathbf{0} &= \mathbf{g}(t_i, \mathbf{X}_i, \mathbf{Z}_i), \quad i = 1, 2, \ldots, s.
\end{aligned}$$

For the algebraic variables $\mathbf{z}$ it is often better to avoid the quadrature step implied by the last step of a usual RK method.

$\rightarrow$ There is no corresponding integration in the DAE!

This makes stiffly accurate methods with $a_{sj} = b_j$ desirable: the constraints are automatically satisfied at the final stage; i.e.,

$$\mathbf{y}_n = \mathbf{Y}_s,$$

and there is no need for the final (quadrature) step.

We then set $\mathbf{x}_n = \mathbf{X}_s$.

As for (non-BDF) multistep methods, there are additional order conditions that such methods must satisfy to have order > 2 for general index-1 and Hessenberg index-2 DAEs.

Methods often exhibit order reduction as they do when solving stiff ODEs.

This is perhaps not surprising given the close relationship between DAEs and stiff ODEs.

A detailed discussion of the theory behind the causes of order reduction is given in the text.

The rooted tree theory of Butcher has also been extended to provide a complete set of necessary and sufficient conditions for certain classes of DAEs, e.g., semi-explicit index-1, general index-1, and Hessenberg index-2 and index-3.

Convergence results for Runge–Kutta methods that are also collocation methods are given in the text.

10.1.4 Practical Considerations

Despite the nice order and convergence results, there remain some practical issues to consider.

• Consistent Initial Conditions

A major practical difference between numerically solving ODEs and DAEs is that the DAE system must be initialized with values that are *consistent* with all the constraints – even the hidden ones!

Sometimes there is not enough information for the DAE to have a solution; sometimes there is too much or the wrong type of information.

Consider a semi-explicit index-1 DAE, and suppose

$$\mathbf{x}(0) = \mathbf{x}_0.$$

This is exactly all we need to theoretically solve the problem: $\mathbf{z}(0) := \mathbf{z}_0$ can be obtained from solving

$$\mathbf{0} = \mathbf{g}(0, \mathbf{x}_0, \mathbf{z}_0),$$

then $\dot{\mathbf{x}}(0)$ can be evaluated, etc.

However, a DAE solver may require a value for $\mathbf{z}_0$ or face a "cold start", and there is no treatment of what to do if there is more than one solution.

Example 1. *Consider the semi-explicit index-1 DAE*

$$\begin{aligned} \dot{u} &= -(u+v)/2 + q_1(t), \\ 0 &= (u-v)/2 - q_2(t). \end{aligned}$$

In this case, we can specify $u(0)$ arbitrarily, and this determines $v(0) = u(0) - 2q_2(0)$.

Hence we can determine $\dot{u}(0)$.

But now defining $u = y_1 + y_2$ *and* $v = y_1 - y_2$ *yields the DAE*

$$\dot{y}_1 + \dot{y}_2 = -y_1 + q_1(t),$$
$$0 = y_2 - q_2(t).$$

This DAE requires $y_2(0) = q_2(0)$ *for consistency, so although we can still arbitrarily specify* $y_1(0)$*, we cannot determine* $\dot{y}_1(0)$ *without using* $\dot{y}_2(0) = \dot{q}(0)$*.*

Of course, the situation gets more complicated for more general index-1 DAEs and higher-index DAEs.

Example 2. *Recall the Hessenberg index-3 DAE for the simple pendulum in Cartesian co-ordinates.*

We note first that $\mathbf{q}(0)$ *cannot be assigned arbitrarily; e.g., given* $q_1(0)$, $q_2(0)$ *is determined (up to a sign) from the constraint on the pendulum length.*

Then from the (hidden) constraint on the velocity level, one component of $\mathbf{v}$ *is determined from the other.*

In other words, the user's specification of $\mathbf{q}(0)$ *and* $\mathbf{v}(0)$ *must be consistent with the constraints on the position and velocity levels.*

These conditions determine $\dot{\mathbf{q}}(0)$ *and hence* $\lambda(0)$.

In general, consistent initialization for high-index systems is usually done on a case-by-case basis.

- Ill-Conditioned Newton Iteration Matrix

Recall, for explicit ODEs, the Newton iteration matrix tends to $\mathbf{I}$ as $\Delta t_n \to 0$.

For index-1 and Hessenberg DAEs, it can be shown that the condition number of the Newton iteration matrix is $\mathcal{O}((\Delta t_n)^{-d})$, where d is the index.

For example, the Newton iteration matrix for backward Euler applied to the semi-explicit index-1 DAE is

$$\begin{pmatrix} (\Delta t_n)^{-1}\mathbf{I} - \mathbf{f_x} & -\mathbf{f_z} \\ -\mathbf{g_x} & -\mathbf{g_z} \end{pmatrix}.$$

The condition number of this matrix is $\mathcal{O}((\Delta t)_n^{-1})$.

For small Δt_n, the Newton iteration might fail.

These bad condition numbers can be ameliorated to some extent by scaling the constraints (for semi-explicit index-1 DAEs) and the algebraic variables (for Hessenberg index-2 DAEs).

- Error estimation for index-2 DAEs

Recall, modern BDF codes estimate the local errors using a weighted norm of a divided difference of $\mathbf{y}$.

This error estimate still works for fully implicit index-1 DAEs, *but it fails for index-2 problems!*

See Example 10.6 in the text, where it is shown that for a simple index-2 DAE, the standard way of estimating the local truncation error by divided differences yields estimates that approach 0 as $\Delta t_n \to 0$ only for the differential variables, not for the algebraic variables!

This will result in repeated rejected steps.

This problem can be fixed by not including the algebraic variables (in particular the index-2 variables) in the error estimate.

It has been shown that this strategy is safe: it turns out the lower-index variables are still accurate (the estimate only thinks they are not).

Note: The algebraic variables should not be removed from the Newton convergence test!

10.1.5 Specialized RK methods for Hessenberg Index-2 DAEs

We now focus on Runge–Kutta methods for Hessenberg index-2 DAEs in order to take advantage of the special structure (i.e., the pure index-2 nature of the algebraic variables) of these problems.

- Projected Runge–Kutta methods

Recall, Radau methods are not preferred for BVPs because they are not symmetric.

Gauss collocation methods were successful for BVODEs, but because they do not have stiff decay, they suffer from severe order reduction when applied in a straightforward way to Hessenberg index-2 DAEs.

e.g., the midpoint method is only $\mathcal{O}(1)$ accurate for the index-2 variable $\mathbf{z}$.

Additional difficulties include potential instability and no nice local error expansion.

Fortunately, *all these problems can be solved by projecting onto the constraint manifold at the end of each step!*

Note: This may mean that the piecewise polynomials approximating $\mathbf{x}(t)$ and $\mathbf{z}(t)$ may become discontinuous at the t_n.

Suppose we take a step from $\mathbf{x}_{n-1},\ \mathbf{z}_{n-1}$ using an implicit RK method.

The idea behind projection is to perturb the resulting $\mathbf{x}_n, \mathbf{z}_n$ to satisfy

$$\begin{aligned} \hat{\mathbf{x}}_n &= \mathbf{x}_n + \mathbf{f}_{\mathbf{z}}(t_n, \mathbf{x}_n, \mathbf{z}_n)\boldsymbol{\lambda}_n, \\ \mathbf{0} &= \mathbf{g}(t_n, \hat{\mathbf{x}}_n), \end{aligned}$$

and take $\mathbf{x}_n = \hat{\mathbf{x}}_n$ as the numerical solution.

Note: The $\boldsymbol{\lambda}_n$ are not stored.

For a method with stiff decay, the constraint is already satisfied, so there is no need to project.

For a collocation method without stiff decay, e.g., Gauss collocation, the projection restores all the nice features that the method had for BVODEs, in particular superconvergence for $\mathbf{x}$ at the mesh points.

A postprocessing step allows $\mathbf{z}$ to be determined from $\mathbf{x}$ to the same order of accuracy.

• Half-explicit Runge–Kutta methods

A fully implicit discretization is overkill if the differential equations are non-stiff; i.e., the stiffness enters only via the constraints.

Many models for mechanical systems are like this.

One approach for taking advantage of this is via *half-explicit RK methods*.

For a semi-explicit DAE, the half-explicit RK method is defined by

$$\begin{aligned}
\mathbf{X}_i &= \mathbf{x}_{n-1} + \Delta t_n \sum_{j=1}^{i-1} a_{ij} \mathbf{f}(t_{nj}, \mathbf{X}_j, \mathbf{Z}_j), \\
\mathbf{0} &= \mathbf{g}(t_{ni}, \mathbf{X}_i, \mathbf{Z}_i), \quad i = 1, 2, \ldots, s, \\
\mathbf{x}_n &= \mathbf{x}_{n-1} + \Delta t_n \sum_{i=1}^{s} b_i \mathbf{f}(t_{ni}, \mathbf{X}_i, \mathbf{Z}_i), \\
\mathbf{0} &= \mathbf{g}(t_n, \mathbf{x}_n, \mathbf{z}_n);
\end{aligned}$$

i.e., at each stage i, the $\mathbf{X}_i$ are evaluated explicitly, and the $\mathbf{Z}_i$ are obtained by solving a nonlinear system.

For semi-explicit index-1 DAEs, the order of accuracy is the same as for ODEs.

$\rightarrow$ these methods are in fact not very different from applying an explicit RK method to

$$\dot{\mathbf{x}} = \mathbf{f}(t, \mathbf{x}, \mathbf{z}(\mathbf{x})).$$

There is generally order reduction when these methods are applied to semi-explicit Hessenberg systems of index 2, but techniques have been developed to address this as well.

10.2 Methods for ODEs on Manifolds

The numerical solution of ODEs on manifolds forms an interesting problem class in its own right; as discussed, DAEs are not the only source of such problems.

Nonetheless, this provides a useful perspective for the numerical solution of DAEs.

To recall, consider the nonlinear IVP

$$\dot{\mathbf{x}} = \hat{\mathbf{f}}(\mathbf{x}), \qquad \mathbf{x}(t_0) = \mathbf{x}_0,$$

such that there exists an invariant set $\mathcal{M}$ defined by the algebraic equations

$$\mathbf{0} = \mathbf{h}(\mathbf{x});$$

i.e.,

$$\mathbf{h}(\mathbf{x}_0) = \mathbf{0} \implies \mathbf{h}(\mathbf{x}(t)) = \mathbf{0} \ \forall t.$$

Note 1. *From the perspective of dynamical systems, this could be forward or backward in time.*

Approaches to solve this problem include

- solving the stabilized ODE

$$\dot{\mathbf{x}} = \hat{\mathbf{f}}(\mathbf{x}) - \gamma \mathbf{F}(\mathbf{x})\mathbf{h}(\mathbf{x}),$$

  by choosing appropriate $\mathbf{F}(\mathbf{x})$ and γ;

- using some sort of stabilization at the end of each step to bring $\mathbf{x}_n$ closer to satisfying $\mathbf{h}(\mathbf{x}_n) = \mathbf{0}$ (post-stabilization or co-ordinate projection);

- finding a numerical method that automatically satisfies $\mathbf{h}(\mathbf{x}) = \mathbf{0}$; this is possible when $\mathbf{h}(\mathbf{x})$ is (at most) a quadratic function.

Of these, we now discuss post-stabilization and co-ordinate projection.

10.2.1 Stabilization of the Discrete Dynamical System

If the ODE is not stiff, then we would like to use a non-stiff method for its solution.

In order to enforce the constraint, we apply a stabilization at the end of each step.

Suppose we use our favourite method to (tentatively) advance the solution of the ODE as

$$\tilde{\mathbf{x}}_n = \phi^{\hat{\mathbf{f}}}_{\Delta t_n}(\mathbf{x}_{n-1}).$$

then post-stabilize $\tilde{\mathbf{x}}_n$ to obtain

$$\mathbf{x}_n = \tilde{\mathbf{x}}_n - \mathbf{F}(\tilde{\mathbf{x}}_n)\mathbf{h}(\tilde{\mathbf{x}}_n).$$

For post-stabilization to be effective, we must choose $\mathbf{F}$ such that

$$\|\mathbf{I} - \mathbf{HF}\| \leq \rho < 1,$$

where $\mathbf{H} = \mathbf{h}_{\mathbf{x}}$.

With sufficient smoothness near $\mathcal{M}$ and $\rho = \mathcal{O}(\Delta t_n)$, the following will apply to an ODE method of order p.

- Stabilization does not change the method's order of convergence:

$$\mathbf{x}(t_n) - \mathbf{x}_n = \mathcal{O}((\Delta t_n)^p).$$

- There exists a constant K that depends only on local solution properties such that

$$\|\mathbf{h}(\mathbf{x}_n)\| \leq K(\rho(\Delta t_n)^{p+1} + (\Delta t_n)^{2(p+1)}).$$

- $$\|\mathbf{h}(\mathbf{x}_n)\| = \mathcal{O}((\Delta t_n)^{2(p+1)}), \quad \text{if } \mathbf{HF} = \mathbf{I}.$$

Example 10.8 of the text shows some of these results.

A closely related stabilization method is the so-called co-ordinate projection method.

In this case, after taking the tentative step with your favourite numerical method to $\tilde{\mathbf{x}}_n$, we determine $\mathbf{x}_n$ that minimizes $\|\mathbf{x}_n - \tilde{\mathbf{x}}_n\|_2$ such that $\mathbf{h}(\mathbf{x}_n) = \mathbf{0}$.

This is a constrained least-squares problem.

It turns out that performing a post-stabilization with $\mathbf{F} = \mathbf{H}^T(\mathbf{H}\mathbf{H}^T)^{-1}$ (which satisfies $\mathbf{H}\mathbf{F} = \mathbf{I}$) corresponds to one Newton step for the solution to this minimization problem.

So for Δt_n sufficiently small, the two methods are essentially indistinguishable.

Note that $\|\mathbf{h}(\mathbf{x}_n)\| = \mathcal{O}((\Delta t_n)^{2(p+1)})$, so the invariant is well satisfied and remains so for future steps without the need for more than one iteration per step.

However, post-stabilization has the advantage of being more general (and hence more flexible in choosing $\mathbf{F}$), so it can be more efficient.

10.2.2 Choosing the Stabilization Matrix

Post-stabilization steps are more effective as $\|\mathbf{I} - \mathbf{HF}\|$ is made smaller.

The choice $\mathbf{F} = \mathbf{D}(\mathbf{HD})^{-1}$ achieves the optimum value of $\|\mathbf{I} - \mathbf{HF}\| = 0$.

However, a given choice of $\mathbf{D}$ (and hence $\mathbf{F}$) may be (too) expensive to use.

For example, for the Euler–Lagrange equations of motion for multi-body systems,

$$\mathbf{H} = \begin{pmatrix} \mathbf{G} & \mathbf{0} \\ \frac{\partial(\mathbf{Gv})}{\partial \mathbf{q}} & \mathbf{G} \end{pmatrix},$$

which involves using the evil matrix $\partial(\mathbf{Gv})/\partial\mathbf{q}$.

Example 3. *Here is an algorithm for post-stabilization applied to multi-body systems.*

1. *Use your favourite ODE integrator to tentatively advance the system*

$$\begin{aligned}\dot{\mathbf{q}} &= \mathbf{v},\\ \mathbf{M}(\mathbf{q})\dot{\mathbf{v}} &= \mathbf{f}(\mathbf{q},\mathbf{v}) - \mathbf{G}^T(\mathbf{q})\boldsymbol{\lambda},\\ \mathbf{0} &= \mathbf{G}(\mathbf{q})\dot{\mathbf{v}} + \frac{\partial(\mathbf{G}\mathbf{v})}{\partial \mathbf{q}}\mathbf{v},\end{aligned}$$

   from $(\mathbf{q}_{n-1}, \mathbf{v}_{n-1})$ *at time* $t = t_{n-1}$ *to* $(\tilde{\mathbf{q}}_n, \tilde{\mathbf{v}}_n)$ *at time* $t = t_n$.

2. *Let*

$$\mathbf{F} = \begin{pmatrix} \mathbf{B}(\mathbf{G}\mathbf{B})^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{B}(\mathbf{G}\mathbf{B})^{-1} \end{pmatrix},$$

   where $\mathbf{B} = \mathbf{M}^{-1}\mathbf{G}^T$.

   Post-stabilize:

$$\begin{pmatrix} \hat{\mathbf{q}}_n \\ \hat{\mathbf{v}}_n \end{pmatrix} = \begin{pmatrix} \tilde{\mathbf{q}}_n \\ \tilde{\mathbf{v}}_n \end{pmatrix} - \mathbf{F}(\tilde{\mathbf{q}}_n, \tilde{\mathbf{v}}_n)\mathbf{h}(\tilde{\mathbf{q}}_n, \tilde{\mathbf{v}}_n).$$

Note 2.

$$\mathbf{HF} = \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{L} & \mathbf{I} \end{pmatrix}, \quad \textit{where } \mathbf{L} = \frac{\partial(\mathbf{Gv})}{\partial \mathbf{q}} \mathbf{B}(\mathbf{GB})^{-1};$$

thus $(\mathbf{I} - \mathbf{HF}) \neq \mathbf{0}$, *but* $(\mathbf{I} - \mathbf{HF})^2 = \mathbf{0}$*!*

So post-stabilizing with $\mathbf{F}$ *twice is an excellent idea.*

Note 3. *The decompositions required in applying* $\mathbf{F}$ *can be stored so that the second post-stabilization is essentially free.*

Note 4. $\mathbf{F}$ *can be frozen over multiple steps.*

3. *Repeat post-stabilization step:*

$$\begin{pmatrix} \mathbf{q}_n \\ \mathbf{v}_n \end{pmatrix} = \begin{pmatrix} \hat{\mathbf{q}}_n \\ \hat{\mathbf{v}}_n \end{pmatrix} - \mathbf{F}(\hat{\mathbf{q}}_n, \hat{\mathbf{v}}_n)\mathbf{h}(\hat{\mathbf{q}}_n, \hat{\mathbf{v}}_n).$$

Note 5. *A sometimes-better (but generally more expensive)* $\mathbf{F}$ *is*

$$\begin{pmatrix} \mathbf{G}^T(\mathbf{G}\mathbf{G}^T)^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{G}^T(\mathbf{G}\mathbf{G}^T)^{-1} \end{pmatrix}.$$

Note 6. *If the multi-body system has non-holonomic constraints, then the DAE is index 2 and only one stabilization using* $\mathbf{F}$ *is needed per step.*

See Example 10.9 in the text to see some results of this algorithm in action.