Tridiagonal Matrix Full Solve Times

Factor Time (seconds)

System Size: n
A Computational Mechanics Example: Spring-Mass Equilibria and Time Scales

Relevant forces:

- Gravity:
  \[ f_{g,i} = g m_i \]

- Spring forces:
  \[
  \begin{align*}
  f_{s,1} &= -k_1 d_1 + k_2 (d_2 - d_1) \\
  f_{s,2} &= -k_2 (d_2 - d_1) + k_3 (d_3 - d_2) \\
  f_{s,3} &= -k_3 (d_3 - d_2) + k_4 d_3
  \end{align*}
  \]
**Fixed-End Case**

**Equilibrium:** Net force = 0.

\[ f_{g,i} + f_{s,i} = 0 \iff -f_{s,i} = f_{g,i}, \quad i = 1, \ldots, 3. \]

\[
\begin{bmatrix}
(k_1 + k_2) & -k_2 \\
-k_2 & (k_2 + k_3) & -k_3 \\
-k_3 & (k_3 + k_4)
\end{bmatrix}
\begin{bmatrix}
d_1 \\
d_2 \\
d_3
\end{bmatrix}
= 
\begin{bmatrix}
m_{1g} \\
m_{2g} \\
m_{3g}
\end{bmatrix}
\]
Free-End Case

• Equivalent to fixed-end case with $k_4 = 0$:

\[
\begin{bmatrix}
(k_1 + k_2) & -k_2 \\
-k_2 & (k_2 + k_3) & -k_3 \\
-k_3 & k_3 \\
\end{bmatrix}
\begin{pmatrix}
d_1 \\
d_2 \\
d_3 \\
\end{pmatrix}
= 
\begin{pmatrix}
m_1 g \\
m_2 g \\
m_3 g \\
\end{pmatrix}.
\]

• If $k_i = k$ and $m_i = m$, we have

\[
\begin{bmatrix}
2 & -1 \\
-1 & 2 & -1 \\
-1 & 1 \\
\end{bmatrix}
\begin{pmatrix}
d_1 \\
d_2 \\
d_3 \\
\end{pmatrix}
= \frac{mg}{k}
\begin{pmatrix}
1 \\
1 \\
1 \\
\end{pmatrix}.
\]
Some Comments on Solving the Equilibrium Problem

- Last time, we argued in favor of using `sparse(A)` in matlab for our tridiagonal matrix $A$.

- We also need to emphasize the importance of

  \[ d = A\backslash f \]

  instead of

  \[ d = \text{inv}(A) \times f \]

- The reason is that $A^{-1}$ is full, with $n^2$ nonzeros.

- To see this, we’re going to revisit matrix-vector products of the form $\underline{w} = A\underline{u}$, and some physics.

- The two combined will shed light on why `inv(A)` is a very bad idea in our context.
A Computational Mechanics Example: Spring-Mass Motion

- Newton’s 2nd Law: \( ma = F_{\text{net}}. \)
  - Acceleration: \( a_i = \ddot{d}_i. \)
  - Net force: \( f_i = f_{s,i} + f_{g,i} \)

- Linear System.

For \( k=\text{constant}, \) fixed-end case:

\[
\begin{bmatrix}
m_1 \\
m_2 \\
\vdots \\
m_n
\end{bmatrix}
\begin{bmatrix}
\ddot{d}_1 \\
\ddot{d}_2 \\
\ddots \\
\ddot{d}_n
\end{bmatrix} = -k
\begin{bmatrix}
2 & -1 & & \\
-1 & 2 & -1 & \\
& & \ddots & \\
& & & -1 & 2
\end{bmatrix}
\begin{bmatrix}
d_1 \\
d_2 \\
\vdots \\
d_n
\end{bmatrix} + g
\begin{bmatrix}
m_1 \\
m_2 \\
\vdots \\
m_n
\end{bmatrix}
\]
Here, $\underline{d}(t)$ is the vector of unknown displacements,

$$
\underline{d}(t) := \begin{pmatrix}
  d_1(t) \\
  d_2(t) \\
  \vdots \\
  d_n(t)
\end{pmatrix}.
$$

• For general $k$:

$$
M\ddot{\underline{d}} = -A\underline{d}(t) + \underline{f}.
$$

• Initial Conditions.

Displacement: $\underline{d}(t = 0) = \underline{d}^0$

Velocity: $\dot{\underline{d}}(t = 0) = \dot{\underline{d}}^0$. 
Conversion to a Homogeneous System

• Starting with

\[ M \ddot{d} = -Ad(t) + f. \]

• Suppose \( d^\infty \) satisfies the (steady-state) equilibrium condition,

\[ Ad^\infty = f. \]

• Let \( \delta := d - d^\infty \) be the departure from this equilibrium condition.
• Then

\[ \dot{\delta} = \dot{d} \]

\[ \ddot{\delta} = \ddot{d} \]

and

\[ M\ddot{\delta} = M\ddot{d} = -A (\delta + d^\infty) + f. \]

\[ = -A\delta. \]

• Initial conditions are:

\[ \delta(0) = d^0 - d^\infty \]

\[ \dot{\delta}(0) = \dot{d}^0. \]
• We now have a homogeneous initial-value problem (IVP):

$$M\ddot{\delta} = -A\delta + \text{I.C.s.}$$

• It is second-order in time.

• To solve this problem numerically, we rewrite it as a first-order system.

• Let

$$v := \ddot{\delta}$$

$$q := \begin{pmatrix} \delta \\ v \end{pmatrix}$$

$$\dot{q} = \begin{pmatrix} \dot{\delta} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} \ddot{\delta} \\ \ddot{\delta} \end{pmatrix}.$$
• With these variables, we rewrite our 2nd-order ODE as a first-order system as follows.

\[
\begin{bmatrix}
    I & M \\
    0 & A
\end{bmatrix}
\begin{bmatrix}
    \dot{\delta} \\
    \dot{v}
\end{bmatrix}
=
\begin{bmatrix}
    0 & I \\
    -A & 0
\end{bmatrix}
\begin{bmatrix}
    \delta \\
    v
\end{bmatrix}.
\]

• In terms of \(q(t)\), we have,

\[
\dot{q} = \begin{bmatrix}
    I \\
    M^{-1}
\end{bmatrix}
\begin{bmatrix}
    0 & I \\
    -A & 0
\end{bmatrix}
q
\]

\[
= \begin{bmatrix}
    0 & I \\
    -M^{-1}A & 0
\end{bmatrix}
q
\]

\[
= Lq
\]

\[
dq/dt = Lq + \text{I.C.s.}
\]
• Simplest timestepper – \textit{Euler forward} (EF).
  (NB: EF is actually a very poor choice for this problem.
  It comes with no endorsement.
  It is, however, easy to understand.)

• Using Taylor series, we replace the time derivative by a finite difference,

\[
\left. \frac{dq}{dt} \right|_{t_{m-1}} = \frac{q^m - q^{m-1}}{\Delta t} + O(\Delta t) = Lq^{m-1}.
\]

• Rearranging to solve for the unknown $q^m$ at time $t^m$,

\[
q^m = q^{m-1} + \Delta t Lq^{m-1} + O(\Delta t^2).
\]
• For reasons we’ll see later in the course, the unknown $O(\Delta t^2)$ residual is called the local truncation error (LTE).

• It is typically one order higher than the global truncation error (GTE), which for Euler forward is $O(\Delta t)$.

• The GTE is the error you observe in your final result, after integrating out to fixed time $t = T$. (Here, $T$ is presumably prescribed by the user or problem definition.)

• To obtain Euler forward, we drop the $O(\Delta t^2)$ residual to yield

$$q^m = q^{m-1} + \Delta t L q^{m-1}.$$

*This is our timestepping algorithm.*

• When coupled with the initial condition $q^0$, EF provides a simple way (perhaps the simplest?) to advance from time level $t^{m-1}$ to $t^m$.

(Note: If $\Delta t$ is constant, then $t^m = m\Delta t$. We will generally assume $\Delta t$=constant, unless otherwise stated.)
How to choose $\Delta t$?

Possible strategies / considerations:

- Trial and error. (Why not?)
- A bit of analysis.
  
  1. How large is $T$?
  2. How does accuracy (at time $T$) depend on $\Delta t$?
  3. Does $\Delta t$ impact the stability of the algorithm?
To address these questions, let’s start with a single mass, free-end.

\[ m\ddot{d} = -kd, \quad d(0) = 1, \quad \dot{d}(0) = 0, \quad \text{(say).} \]

- Mechanical insight: the spring oscillates.

- Mathematics (ODE): consider a solution of the form \( d(t) = ae^{st} \).

\[
\begin{align*}
\dot{d} &= ase^{st} = asd \\
\ddot{d} &= as^2e^{st} = as^2d = -\frac{k}{m}d.
\end{align*}
\]

- Thus,

\[ s = \pm \sqrt{-\frac{k}{m}} = \pm i\sqrt{\frac{k}{m}} = \pm i\omega. \]

where \( i := \sqrt{-1} \) and \( \omega := \sqrt{k/m} \).
• Since our equation is linear (in $d$) and we have two roots, $s_1$ and $s_2$ (our equation is second order), we can use superposition to add these two fundamental solutions,

$$d = a_1 e^{s_1 t} + a_2 e^{s_2 t}$$

$$= a_1 e^{i\omega t} + a_2 e^{-i\omega t}.$$

• At $t = 0$ we have:

$$d = a_1 + a_2 = 1$$  

$$\dot{d} = i\omega a_1 - i\omega a_2 = 0,$$

from which we deduce $a_1 = a_2 = \frac{1}{2}$.

• Thus, the solution satisfying the differential equation and its initial condition is

$$d = \frac{1}{2} (e^{i\omega t} + e^{-i\omega t})$$

$$= \cos \omega t = \cos \frac{2\pi t}{\tau}.$$
Here, we have introduced the period,

\[ \tau := \frac{2\pi}{\omega} = 2\pi \sqrt{\frac{m}{k}}, \]

which is the most important time scale (aside, perhaps, from \( T \)).

Typically, we need \( \frac{\Delta t}{\tau} \ll 1 \) for accurate \textit{numerical integration} (timestepping), say, at least

\[ \frac{\Delta t}{\tau} \approx \frac{1}{20}, \]

but the actual value depends on the choice of timestepper and the desired accuracy at the final time \( T \).
• What happens if there is more than one frequency?

• Possible options:
  – Choose $\Delta t < \tau_{\text{min}}$.

  – Use an implicit method that is stable for $\Delta t > \tau_{\text{min}}$, where $\tau_{\text{min}}$ is the smallest period that is relevant to your problem.

  – Use a model that eliminates the fast timescales (i.e., corresponding to $\tau < \tau_{\text{min}}$).

• Let’s look at some examples of timestepping options and slightly more complex systems.
• First, a quick summary:
  
  – We identified a relevant solution to the problem and the corresponding physical timescale, \( \tau \).

  – Our numerical solution has to be able to accurately represent (even just in a visual sense) this solution, which means \( \Delta t < \tau \).

  – In the case of multiple timescales, we typically are interested in some of them, say the ones having \( \tau > \tau_{\text{min}} \), where \( \tau_{\text{min}} \) is a value chosen by the engineer.
**Relationship of $\Delta t$ to EF Scheme.**

- Recall, our EF scheme is applied to the 1st-order equation,
  
  \[
  \frac{dq}{dt} = Lq = \begin{bmatrix} 0 & 1 \\ -k/m & 0 \end{bmatrix} \begin{pmatrix} d \\ v \end{pmatrix}, \quad \begin{cases} d^0 = 1 \\ v^0 = 0 \end{cases}
  \]

- **Q:** How does $\omega$ relate to this system?

- **A:** Eigenvalues of $L$.

- Assume $L$ has a complete set of (possibly complex) eigenvectors $z_k, k = 1, \ldots, n_L =: n$ s.t.

  \[
  Lz_k = \lambda_k z_k
  \]

  \[
  q = \sum_{k=1}^{n} \hat{q}_k z_k = \sum_{k=1}^{n} z_k \hat{q}_k = Z\hat{q}
  \]

  \[
  \hat{q} = Z^{-1} q.
  \]
• Everything is linear (in $q$) so we can rewrite our IVP in terms of $\hat{q}$:

$$\frac{d}{dt}q = \frac{d}{dt} \sum_{k=1}^{n} z_k \hat{q}_k$$

$$= \sum_{k=1}^{n} z_k \frac{d \hat{q}_k}{dt} = Z \frac{d \hat{q}}{dt}.$$ 

$$L_q = L \left( \sum_{k=1}^{n} z_k \hat{q}_k \right)$$

$$= \sum_{k=1}^{n} L z_k \hat{q}_k$$

$$= \sum_{k=1}^{n} \lambda_k z_k \hat{q}_k = \begin{bmatrix} \lambda_1 & & & \lambda_1 \\ & \bar{\Lambda} & & \\ & & \ddots & \\ & & & \lambda_n \end{bmatrix} \hat{q}$$
• Inserting these expansions into the IVP we have,

\[ Z \frac{d \hat{q}}{dt} = Z \Lambda \hat{q} + \text{I.C.} \]

\[ \frac{d \hat{q}}{dt} = \Lambda \hat{q} \]

• Because \( \Lambda \) is diagonal, this system decouples into \( n \) separate equations,

\[ \frac{d \hat{q}_k}{dt} = \lambda_a \hat{q}_k, \quad k = 1, \ldots, n \]

\[ \hat{q}^0 = Z^{-1} q^0 \]

Solution: \( \hat{q}_k = \hat{q}_k^0 e^{\lambda_k t} \).
For our system, with \( a = -k/m \), we have
\[
L = \begin{bmatrix}
0 & 1 \\
-\frac{k}{m} & 0
\end{bmatrix}.
\]

The eigenvalues satisfy
\[
\lambda : \quad L\ddot{z} = \lambda \ddot{z}
\]
\[
(L - I\lambda)\ddot{z} = 0
\]
\[
|L - I\lambda| = 0 \quad \text{(determinant)}.
\]

(Or \([V, D] = \text{eig}(\text{full}(L))\) in matlab. :)

\[
\begin{vmatrix}
-\lambda & 1 \\
a & -\lambda
\end{vmatrix} = \lambda^2 - a
\]

\[
\lambda = \pm \sqrt{a} = \pm \sqrt{-k/m} = \pm i\sqrt{k/m} = \pm i\omega.
\]

So we see that our 1st-order system give the same temporal response as our 2nd-order IVP, as should be expected.

In general, it’s often simpler to look at the eigenvalues of the first-order system, which is what we will primarily do later in the course.
Two timesteppers: Euler Forward and Euler Backward.

- Evaluate at given time point:
  \[
  \text{EF (explicit)} \quad \left. \frac{d\delta}{dt} \right|_{t^{n-1}} = L\delta^{n-1}
  \]

  \[
  \text{EB (implicit)} \quad \left. \frac{d\delta}{dt} \right|_{t^n} = L\delta^n
  \]

- First-order approximation to derivative:
  \[
  \text{EF:} \quad \left. \frac{d\delta}{dt} \right|_{t^{n-1}} = \frac{\delta^n - \delta^{n-1}}{\Delta t} + O(\Delta t)
  \]

  \[
  \text{EB:} \quad \left. \frac{d\delta}{dt} \right|_{t^n} = \frac{\delta^n - \delta^{n-1}}{\Delta t} + O(\Delta t)
  \]
• Update steps:

\[ \delta^n = \delta^{n-1} + \Delta t L \delta^{n-1} = (I + \Delta t L) \delta^{n-1} \]

\[ \text{EB: } \delta^n = \delta^{n-1} + \Delta t L \delta^n \]

\[ (I - \Delta t L) \delta^n = \delta^{n-1} \]

\[ \delta^n = (I - \Delta t L)^{-1} \delta^{n-1}. \]

• Euler backward (a.k.a. *implicit Euler*) requires solving the linear system \((I - \Delta t L)\) at every timestep.

• It is not more accurate than EF.

• It is, however, more *stable*. 
• Consider scalar case, $L = \lambda$:

EF: $\delta^n = (1 + \Delta t \lambda) \delta^{n-1} \approx e^{\lambda \Delta t} \delta^{n-1}$

EB: $\delta^n = \frac{1}{(1 + \Delta t \lambda)} \delta^{n-1} \approx e^{\lambda \Delta t} \delta^{n-1}$.

• What happens for large $|\lambda \Delta t|$ in these cases?

• Let’s look at some examples.