Lecture 40

Rayleigh quotient iteration

Note: Rayleigh quotient obtains (approximation to) eigenvalue when
(approx.) eigenvector is known.

Inverse iteration: obtains eigenvector from eigenvalue.
Rayleigh quotient iteration: combines both ideas.

Idea: update \( \mu \)

Algorithm

1. \( v^{(0)} \): given, \( ||v^{(0)}||_2 = 1 \), \( 2^{(0)} = (v^{(0)})^T A v^{(0)} \) % corresponding Rayleigh quotient

2. for \( k = 1, 2, \ldots \)

3. solve \( (A - 2^{(k-1)} I) w = v^{(k-1)} \) % apply \( (A - 2^{(k-1)} I)^{-1} \)

4. \( v^{(k)} = w / ||w||_2 \)

5. \( 2^{(k)} = (v^{(k)})^T A v^{(k)} \) % apply Rayleigh quotient
If \( v^{(0)} \) is sufficiently close to an eigenvector \( \theta_j \), then

\[
\begin{align*}
\|v^{(k+1)} - (\pm \theta_j)\| &= O\left(\|v^{(k)} - (\pm \theta_j)\|^3\right) \\
\lambda^{(k+1)} - 2\lambda - 1 &= O\left(\lambda^{(k)} - 2\lambda - 1^3\right)
\end{align*}
\]

\[\text{cubic convergence}\]

**Proof:** Omit

\[A = \begin{pmatrix} 2 & 1 & 1 \\ 1 & 3 & 1 \\ 1 & 1 & 4 \end{pmatrix}, \quad \lambda_1 = 5.21431973377, \quad v^{(0)} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \quad \frac{1}{\sqrt{3}}\]

<table>
<thead>
<tr>
<th>( k )</th>
<th>Power method</th>
<th>Shifter Inverse Iteration, ( \mu = 5 )</th>
<th>Rayleigh Quotient Iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5.0</td>
<td>5.0</td>
<td>5.0</td>
</tr>
<tr>
<td>1</td>
<td>5.18181818</td>
<td>5.2143114</td>
<td>5.2143197</td>
</tr>
<tr>
<td>2</td>
<td>5.208192</td>
<td>5.214312617</td>
<td>5.214319743184</td>
</tr>
</tbody>
</table>

\[\text{Note: each iteration of Rayleigh quotient iteration method triples the} \]
\[\# \text{ of digits of accuracy}\]
For more info: see Numerical Linear Algebra by L.T. Trefethen & D. Bau, pg. 202-210

Back to systems of ODEs

Recall: IVP
\[ y' = f(y) \]
\[ y(0) = y_0 \]

or
\[ y' = f(t, y) \]
\[ y(a) = y_0 \]

For scalar equations (IVPs, rather), we discussed
- existence, uniqueness, well-posedness
- Euler, modified Euler, Runge-Kutta (explicit) one-step methods
- backward Euler (implicit)
- explicit/implicit methods
- one-step or multistep methods (more - later)

How can we extend these methods to systems of ODEs?
Define $m$th order system of 1st order IVPs:

\[
\begin{align*}
    u_1' &= f_1(t, u_1, u_2, \ldots, u_m) \\
    u_2' &= f_2(t, u_1, u_2, \ldots, u_m) \\
    & \quad \vdots \\
    u_m' &= f_m(t, u_1, u_2, \ldots, u_m)
\end{align*}
\]

for $a \leq t \leq b$ with

\[
    u_1(a) = \alpha_1, \quad u_2(a) = \alpha_2, \quad \ldots, \quad u_m(a) = \alpha_m
\]

Vector form

\[
\begin{align*}
    \vec{u}' &= \vec{f}(t, \vec{u}), \quad a \leq t \leq b \\
    \vec{u}'(a) &= \vec{\alpha}
\end{align*}
\]

Then apply numerical method to each component of $\vec{u}$.
How to transform an $n^{th}$ order DE into a system of 1st order DEs?

Consider a single $n^{th}$ order DE:

$$y^{(n)} = f(t, y, y', \ldots, y^{(n-1)})$$

Introduce the dependent variables $y_1, y_2, \ldots, y_n$ as follows:

$$y_1 = y, \quad y_2 = y', \quad y_3 = y'', \ldots, \quad y_n = y^{(n-1)}$$

Note:

$$y_1' = y' = y_2, \quad y_2' = y'' = y_3, \ldots, \quad y_n' = y^{(n)}$$

This yields the system:

$$\begin{align*}
y_1' &= y_2 \\
y_2' &= y_3 \\
&\vdots \\
y_{n-1}' &= y_n \\
y_n' &= f(t, y_1, y_2, \ldots, y_n)
\end{align*}$$

n equations of 1st order
explicit), and order accurate, a function evaluation per step:

\[ y_{n+1} = y_n + h \left( \frac{3}{2} f(y_n) - f(y_n) \right) \]

Adams Bashforth

Adams-Bashforth may be needed to find \( y_{n+1} \): If \( p_0 \neq 0 \), the source is implicit (a rootfinding required)

2. If \( p_0 = 0 \), the source is explicit

\( y_{n+1} = y_n + h \sum_{i=0}^{p_0} \frac{p_i}{p_0!} f(y_n+i) \)

\( h = f(y) \)

General \( q \)-step method

\[ y_{n+1} = y_n + \sum_{i=1}^{q+1} \beta_i f(y_{n+1-i}) \]

multi-step methods (for solving IVPs)
Adams-Moulton

\[ u_{n+1} = u_n + \frac{h}{12} \left( 5f(u_{n+1}) + 8f(u_n) - f(u_{n-1}) \right) \]

implicit, 3rd order accurate

Note
1. There are \( k \)-step AB and AM methods.
2. A popular predictor-corrector method uses AB as predictor and AM as corrector.

leap-frog

\[ \frac{u_{n+1} - u_{n-1}}{2h} = f(u_n) \]

\[ u_{n+1} = u_{n-1} + 2h f(u_n) \]

explicit, 2nd order accurate, 1 function evaluation per step
BDF: backward differentiation formula (Gear's method)

\[ \frac{3}{2} u_{n+1} - 2 u_n + \frac{1}{2} u_{n-1} = hf(u_n) \]

explicit, 2nd order accurate, 1 function evaluation per step

Software

adaptive time step: ODE23, ODE45 - Matlab

adaptive order methods