Optimal Control, Guidance and Estimation

Lecture – 3

Review of Numerical Methods

Prof. Radhakant Padhi
Dept. of Aerospace Engineering
Indian Institute of Science - Bangalore

---

**Linear Equations:** Solution Technique

Problem:

\[
AX = b \quad A \text{ is nonsingular, } b \neq 0
\]

\[X = ?\]

Motivation:

\[
\dot{X} = AX + BU
\]

where

\[
X = \begin{bmatrix} X_c \\ X_N \end{bmatrix}
\]

- \(X_c\) : controlled state
- \(X_N\) : uncontrolled state
- \(\text{dim}(X_c) = \text{dim}(U) = m\)

\[
\begin{bmatrix} \dot{X}_c \\ \dot{X}_N \end{bmatrix} = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} X + \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} U
\]
Motivation: Continued

\[
\dot{X}_c = A_1 X + B_1 U
\]

Which gives

\[
U = -B_1^{-1} (A_1 X)
\]

Note: \( B_1 \) is square

This will be the controller necessary to maintain \( X_c \) at steady state.

Solution Technique: Direct Inversion of \( A \)

\[
X = A^{-1} b
\]

- Computation of \( A^{-1} \) involves too many computations, roughly \( n^2 \times n! \) number of operations (very inefficient for large \( n \)).

- This approach also suffers from the problem of sensitivity (ill-conditioning), when \( |A| \to 0 \)

- Round off errors may lead to large inaccuracies
Solution Technique: Gauss Elimination

- Do row operations to reduce the A matrix to an upper triangular form
- Solve the variable from down to top

Example:

\[
\begin{bmatrix}
2 & 1 & 0 \\
1 & 2 & 1 \\
0 & 1 & 1
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3
\end{bmatrix}
= \begin{bmatrix}
1 \\
2 \\
4
\end{bmatrix}
\]

Solution Steps:

Step-I: Multiply row-1 by \(-1/2\) and add to row-2. row-3 keep unchanged, since \(a_{31} = 0\).

\[
\begin{bmatrix}
2 & 1 & 0 \\
0 & 3/2 & 1 \\
0 & 1 & 1
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3
\end{bmatrix}
= \begin{bmatrix}
1 \\
3/2 \\
4
\end{bmatrix}
\]

Step-II: Multiply row-2 by \(-2/3\) and add to row-3

Upper Triangle Matrix

Final Solution

\[
\begin{bmatrix}
x_1 \\
x_2 \\
x_3
\end{bmatrix}
= \begin{bmatrix}
3 \\
-5 \\
9
\end{bmatrix}
\]
**Gauss Elimination**

- The total number of operations needed is \((2/3)n^3\), which is far lesser than computing

  \[ A^{-1} = \frac{\text{adj}(A)}{|A|} \]  

  (which requires \(n^2 \times n!\) operations)

- The Gauss elimination method will encounter potential problems when the pivot elements i.e., diagonal elements become zero, or very close to zero at any stage of elimination.

- In such cases the order of equations can be changed by exchanging rows and the procedure can be continued.

**Nonlinear Algebraic Equations**

**Problem:** \(F(X) = 0\) \(\quad X = ?\)

**Motivation:** Finding the forced equilibrium condition for a nonlinear system to get an appropriate operating point for linearization

\[
\begin{bmatrix}
\dot{X}_c \\
\dot{X}_N
\end{bmatrix} =
\begin{bmatrix}
f_c(X, U) \\
f_N(X, U)
\end{bmatrix}
\]

\(\dim(X_c) = \dim(f_c) = \dim(U)\)

\[
\dot{X}_c = f_c(X_0, U_0)
\]

Solve for \(U_0\) from \(f_c(U) = 0\)
Newton-Raphson Method: Scalar Case

\[ f(x) = 0 \]

Using Taylor series expansion

\[ f(x_i + \Delta x) \approx f(x_i) + \left[ \frac{df}{dx} \right]_{x_i} \Delta x + \cdots + (\text{higher order terms}) \]

\[ \left[ \frac{df}{dx} \right]_{x_i} \Delta x = -f(x_i) \]

\[ x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \]

From the above equation with an initial guess, we can iteratively solve for \( x \) with \( \Delta x < \text{tolerance} \)
Newton-Raphson Method: Multi Variable Case

\[ F(X) = 0 \]

\[ F(X_k + \Delta X_k) \approx F(X_k) + \left[ \frac{\partial F}{\partial X} \right]_{X_k} \Delta X_k \]

\[ A_k \Delta X_k = -F(X_k) \]

Solve for \[ \Delta X_k = -A_k^{-1}F(X_k) \]

Update \[ X_{k+1} = X_k + \Delta X_k \]

Newton-Raphson Method: Algorithm

- Start with guess value \( x_1 \)
- Solve for \( \Delta x_k \)
- Update \( x_{k+1} = x_k + \Delta x_k \) \( (k = 1, 2, \ldots) \)
- Continue until convergence

Convergence Condition

1. Relative Error
   \[ \varepsilon_{rel} \triangleq \left| \frac{x_{k+1} - x_k}{x_{k+1}} \right| < \text{tol}, \ \forall k \]

2. Absolute Error
   \[ \|f(x_k)\| < \text{tol} \]
Example: N-R Method

Question: Find a root of the following equation

\[ f(x) = x^3 - 0.165x^2 + 3.993 \times 10^{-4} = 0 \]

Solution: \( f'(x) = 3x^2 - 0.33 \). Let \( x_0 = 0.02 \). Then

\[
\begin{align*}
x_1 &= x_0 - \frac{f(x_0)}{f'(x_0)} = 0.02 - \frac{3.413 \times 10^{-4}}{-5.4 \times 10^{-3}} = 0.08320 \\
x_2 &= x_1 - \frac{f(x_1)}{f'(x_1)} = 0.08320 - \frac{-1.670 \times 10^{-4}}{-6.689 \times 10^{-3}} = 0.05824 \\
x_3 &= x_2 - \frac{f(x_2)}{f'(x_2)} = 0.05824 - \frac{3.717 \times 10^{-5}}{-9.043 \times 10^{-3}} = 0.06235
\end{align*}
\]

Newton-Raphson Method: Advantages

- If it converges, it converges fast!
  It has “Quadratic convergence” property, i.e.

\[
e_{k+1} = c e_k^2, \quad \text{where} \quad e_k \triangleq (x^* - x_k)
\]

- \( c \) is a constant
- \( x^* \) is the actual root

- Problem: It requires good initial guess in general to converge to the right solution.
Newton-Raphson Method: Limitations

- **Non-convergence at Inflection points**

For a function \( f(x) \) the points where the concavity changes from up-to-down or down-to-up are called *inflection points*.

\[
f(x) = (x - 1)^3 = 0
\]

- **Root Jumping**

Cases where \( f(x) \) is oscillating and has a number of roots

Initial Guess near to one root may produce another root

Example:

\[
f(x) = \sin(x) = 0
\]
Newton-Raphson Method: Limitations

- **Oscillations around local minima or maxima**
  Results may oscillate about the local maximum or minimum without converging on a root but converging on the local maximum or minimum. Eventually, it may lead to division by a number close to zero and may diverge.

\[
f(x) = x^2 + 2
\]

\[f(x)\] has no real roots

\[
f(x) = x^2 + 2
\]

Newton-Raphson Method: Limitations

- **Division by zero**
  If \( f'(x_i) \approx 0 \) at some \( x_i \), \( x_{i+1} \) becomes very large value

\[
f(x) = x^3 - 0.03x^2 + 2.4 \times 10^{-6} = 0
\]
\[
f'(x) = 3x^2 - 0.06x
\]

Even after several iterations there is no convergence!
N-R Method Drawbacks

- $f'(x^*)$ is unbounded

If the derivative of $f(x)$ is unbounded at the root then Newton-Raphson method will not converge.

Exercise: Verify for $f(x) = \sqrt{x}$

---

### Numerical Differentiation

<table>
<thead>
<tr>
<th>Technique Name</th>
<th>Definition</th>
<th>Numerical Approximation</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward difference</td>
<td>$\lim_{\Delta x \to 0} \left[ \frac{f(x + \Delta x) - f(x)}{\Delta x} \right]$</td>
<td>$\frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x}$</td>
<td>$O(\Delta x)$</td>
</tr>
<tr>
<td>Backward difference</td>
<td>$\lim_{\Delta x \to 0} \left[ \frac{f(x) - f(x - \Delta x)}{\Delta x} \right]$</td>
<td>$\frac{f(x_0) - f(x_0 - \Delta x)}{\Delta x}$</td>
<td>$O(\Delta x)$</td>
</tr>
<tr>
<td>Central difference</td>
<td>$\lim_{\Delta x \to 0} \left[ \frac{f(x + \Delta x) - f(x - \Delta x)}{2\Delta x} \right]$</td>
<td>$\frac{f(x_0 + \Delta x) - f(x_0 - \Delta x)}{2\Delta x}$</td>
<td>$O(\Delta x^2)$</td>
</tr>
</tbody>
</table>
Numerical Integration

Trapezoidal Rule:

\[ I \approx \frac{1}{2} \Delta x (f_0 + f_1) + \frac{1}{2} \Delta x (f_1 + f_2) + \cdots \]
\[ + \frac{1}{2} \Delta x (f_{n-2} + f_{n-1}) + \frac{1}{2} \Delta x (f_{n-1} + f_n) \]
\[ = \Delta x \left[ f_0 + 2f_1 + 2f_2 + \cdots + 2f_{n-1} + f_n \right] \]
\[ I \approx \left( \frac{f_0}{2} + f_1 + \cdots + f_{n-1} + \frac{f_n}{2} \right) \Delta x \]

Note: Numerical Differentiation is “Error Amplifying”. Where as Numerical Integration is “Error Smoothing”.
Ordinary Differential Equation (ODE)

\[ f \left( x, \frac{dx(t)}{dt}, \frac{d^2x(t)}{dt^2}, \ldots, \frac{d^nx(t)}{dt^n} \right) = 0 \]

- **Ordinary**: only one independent variable
- **Differential Equation**: unknown functions enter into the equation through its derivatives
- **Order**: highest derivative in \( f \)
- **Degree**: exponent of the highest derivative

**Example**:

\[ \left( \frac{d^3x(t)}{dt^3} \right)^4 - x(t) = 0 \]

degree = 4; order = 3

What Is Solution of ODE??

- A problem involving ODE is not completely specified by its equation
  - ODE has to be supplemented with **boundary conditions**.
- **Initial value problem**: \( x \) is given at some starting value \( t_i \), and it is desired to find at some final points \( t_f \) or at some discrete list of points.
- **Two point boundary value problem**: Boundary conditions are specified at more than one \( t \); typically some of the conditions will be specified at \( t_i \) and some at \( t_f \).
Numerical Solution to Initial Value Problem

\[ \frac{dx(t)}{dt} = f(t, x(t)); \quad x(t_0) = x_0 \]

- A numerical solution to this problem generates sequence of values for the independent variable \( t_1, t_2, \ldots, t_n \) and a corresponding sequence of values of the dependent variable \( x_1, x_2, \ldots, x_n \) so that each \( x_n \) approximates solution at \( t_n \)

\[ x_n \approx x(t_n) \quad n=0,1,2,\ldots, n. \]

Basic Concepts of Numerical Methods to Solve ODEs

\[ \frac{x_{n+1} - x_n}{\Delta t} \approx \text{slope of tangent} \]

We can calculate the tangent slope at any point.

In fact the differential equation

\[ \frac{dx(t)}{dt} = f(t, x(t)) \text{ defines the tangent slope } = f(t, x(t)) \]
Euler’s Method

- Solve $\frac{dx}{dt} = f(t, x)$ with $x(0) = b$

- At start of time step

\[ \frac{x_{n+1} - x_n}{\Delta t} \approx f(t_n, x_n) \]  \hspace{1cm} \text{Forward difference}

Rearranging

\[ x_{n+1} = x_n + \Delta t f_n \]

Start with initial conditions $t_0=0; \ x_0=b$

Euler Integration: Useful Comments

- Euler integration has error of the order of $(\Delta t)^2$

- Small step size $\Delta t$ may be needed for good accuracy. This is in conflict with the computational load advantage.

- Lesser computational load
Runge-Kutta Fourth Order Method

\[ x_{i+1} = x_i + \frac{\Delta t}{6} \left( k_1 + 2k_2 + 2k_3 + k_4 \right) \]

where

\[ k_1 = f \left( t_i, x_i \right) \]
\[ k_2 = f \left( t_i + \frac{1}{2} \Delta t, x_i + \frac{1}{2} k_1 \Delta t \right) \]
\[ k_3 = f \left( t_i + \frac{1}{2} \Delta t, x_i + \frac{1}{2} k_2 \Delta t \right) \]
\[ k_4 = f \left( t_i + \Delta t, x_i + k_3 \Delta t \right) \]

In each step the derivative is calculated at four points, once at the initial point, twice at trial mid points and once at trial end point.

Runge-Kutta Algorithm

- Error \[ \theta(\{\Delta t\}^3) \]

- The method uses a 4th order power series approximation to come up with this algorithm. Hence, the algorithm is called RK-4 method.
Thanks for the Attention....!!

Questions ... ??