Lecture 02: Sources of Error and basic error analysis

Outline

Sources of Error
Model/Data Error
Truncation Error
Floating Point Error

Basic definitions of Error Analysis
Truncation Error and Taylor's Theorem
Examples e^x, 1/x
Propagation of Errors and Big O notation
Evaluation of Polynomials and Horner's method

Sources of Error

1) "Model Error": errors in model formulation or choice
e.g. Lotka-Volterra is a poor model for predator-prey
interaction (fractional rabbits, no extinctions)
1a) "Data Error": inaccuracies or uncertainties in model
parameters

   We won't really talk about Model error in this course but it is
   often the most serious source of error and difficult to control.

2) "Truncation Error": e.g. Errors arising from approximating a
function with a simpler function e.g. sin(x)≈ x

3) "Floating Point Error": Errors arising from approximating real
numbers with finite-precision numbers

   In general: All of these errors can contribute to the total error in
   an approximate numerical method (and can be hard to properly
   propagate through a "real problem").
Basic Definitions of Error Analysis

Given a true solution \( f \) and an approximate solution \( \hat{f} \) we define

Absolute error:

Relative error:

Decimal precision \( p \) (number of significant digits):

Example:

\( f = \exp(1) = 2.718281828459046 \quad \hat{f} = 2.71 \)

\( \hat{f} = 2.711345, \hat{f} = 2.74 \)
**Truncation Error and Taylor's Theorem**

**Taylor's Theorem:** Let $f(x)$ be a function $C^{n+1}[a,b]$ and $x_0 \in [a,b]$. Then for every $x \in (a,b)$ there exists a number $c = c(x)$ that lies between $x_0$ and $x$ such that

$$f(x) = T_N(x) + R_N(x)$$

where

$$T_N(x) = \sum_{n=0}^{N} \frac{f^{(n)}(x_0)(x-x_0)^n}{n!}$$

and

$$R_N(x) = \frac{f^{n+1}(c)(x-x_0)^{n+1}}{(n+1)!}$$

---

**Example #1:** $f(x) = e^x$, $x_0 = 0$

1) approximate $f(x) = T_N(x)$
Example #2: \( f(x) = \frac{1}{x} \quad x_0 = 1 \)

1) approximate \( f(x) = T_2(x) \)

Rules of error propagation

in "Big O" notation let

\[
\begin{align*}
  f(h) &= p(h) + O(h^n) \\
  g(h) &= q(h) + O(h^m)
\end{align*}
\]

and \( r = \min(n,m) \) then:
Horner's Method for Evaluation of Polynomials

Given \( P_N(x) = a_0 + a_1x + a_2x^2 + \ldots + a_Nx^N \)

or \( = p_1x^Np_2x^{N-1}p_3x^{N-2} + \ldots + p_{N+1} \)

what is the best way to evaluate \( P_N \) for any value of \( x \)?

Consider \( P_3(x) \) written two ways

\[ P_3(x) = p_1x^3 + p_2x^2 + p_3x + p_4 \]

and as "nested iteration"

\[ P_3(x) = \]

Operation counts:

Horner's Method for Evaluation of Polynomials

Our first algorithm

Evaluate \( y = P(x) \)

given \( p = \{ p_1, p_2, p_3, \ldots, p_{N+1} \} \) and \( x \)

initialize:
Horner's Method for Evaluation of Polynomials

Our first matlab function

```matlab
function  y  = hornersPoly(p,x)
% hornersPoly - evaluates Polynomials using horners rule
% y = hornersPoly(p,x)
%
% p: - vector of polynomial coefficients such that
% y(x) = P(1)x^n + P(2)x^(n-1) + ... + P(n+1)
% x: - values where polynomial is to be evaluated
% y: - outputs y(x)

global STUDENT_ID STUDENT_NAME;
STUDENT_ID = 'mws6';
STUDENT_NAME = 'Marc Spiegelman';

y=p(1);
for i=2:length(p)
    y = y*x+p(i);
end
```

Horner's Method for Evaluation of Polynomials

Our second matlab function - use vectorized arguments

```matlab
function  y  = hornersPolyVec(p,x)
% hornersPolyVec - evaluates Polynomials using horner's rule (vectorized arguments)
% y = hornersPolyVec(p,x)
%
% p: - vector of polynomial coefficients such that
% y(x) = P(1)x^n + P(2)x^(n-1) + ... + P(n+1)
% x: - vector of values where polynomial is to be evaluated
% y: - vector of outputs y(x)

global STUDENT_ID STUDENT_NAME;
STUDENT_ID = 'mws6';
STUDENT_NAME = 'Marc Spiegelman';

y=zeros(size(x));
y(1)=p(1);
for i=2:length(p)
    y = y.*x+p(i);
end
```

This function is algorithmically equivalent to Matlab's polyval function (help polyval)
Limits of Truncation error: Floating point error