

Lecture 11: Numerical Differentiation

Finite Difference Formula: comes from a simple approximation based on computing the slope between two points $(t, f(t))$, $(t + \Delta t, f(t + \Delta t))$, that is the secant line

$$\frac{f(t + \Delta t) - f(t)}{\Delta t}$$

Taking the limit of the secant line as $\Delta t \rightarrow 0$ we get the definition of the derivative of f at t .

$$\frac{df(t)}{dt} = \lim_{\Delta t \rightarrow 0} \frac{f(t + \Delta t) - f(t)}{\Delta t}$$

If we choose Δt to be a fixed non zero value then we can construct a first order forward difference approximation to the derivative:

$$\frac{df(t)}{dt} = \frac{f(t + \Delta t) - f(t)}{\Delta t} + E(f(t), \Delta t)$$

That is, the difference $(f(t + \Delta t) - f(t))$ divided by Δt approximates the derivative when Δt is small.

The total error E is the combined error from truncation and roundoff:

$$E = E_{round} + E_{trunc}$$

First we obtain the truncation error E_{trunc} from Taylor's theorem, first by taking a Taylor expansion (with the assumption that f is differentiable over the domain of consideration, that is all the derivatives we are considering within this context exist):

$$f(t + \Delta t) = f(t) + \Delta t \frac{df(t)}{dt} + \frac{\Delta t^2}{2!} \frac{d^2 f(t)}{dt^2} + \frac{\Delta t^3}{3!} \frac{d^3 f(t)}{dt^3}$$

This is recognized as the Taylor series expansion of $f(t)$ about t .

A useful alternate form of this equation is derived using Taylor's theorem that guarantees that there exists some value c for which the following is true:

$$f(t + \Delta t) = f(t) + \Delta t \frac{df(t)}{dt} + \frac{\Delta t^2}{2!} \frac{d^2 f(c)}{dt^2}$$

where $t - \Delta t \leq c \leq t + \Delta t$.

Subtract $f(t)$ from both sides and divide by Δt

$$\frac{f(t + \Delta t) - f(t)}{\Delta t} = \frac{df(t)}{dt} + \frac{\Delta t}{2!} \frac{d^2 f(c)}{dt^2}$$

Despite we do not have the information of what c is, it is enough for us to be able to put a bound on the truncation error, that is the truncation error is thus of order Δt .

Rearranging

$$\frac{df(t)}{dt} = \frac{f(t + \Delta t) - f(t)}{\Delta t} - \frac{\Delta t}{2!} \frac{d^2 f(c)}{dt^2}$$

Now so far we have assumed that $f(t)$ can be calculated exactly. Once we evaluate expressions for the derivatives round-off error occurs so that

$$f(t) = F(t) + e(t)$$

where $F(t)$ is the approximated value given by the computer and $e(t)$ measures the error from the value of $f(t)$. Thus the round off error for for the term $\frac{f(t+\Delta t)-f(t)}{\Delta t}$ can be written

$$\frac{e(t + \Delta t) - e(t)}{\Delta t}$$

Thus the total error $E = E_{round} + E_{trunc}$ is given by

$$E = \frac{e(t + \Delta t) - e(t)}{\Delta t} - \frac{\Delta t^2}{2} \frac{d^2 f(c)}{dt^2}$$

In order to obtain a maximum size of the error we bound the maximum value of the round-off and the derivative: (question to class: why would we want to find the maximum size of the error?)

$$|e(t + \Delta t)| \leq e_r$$

$$|-e(t)| \leq e_r$$

$$M = \max_{c \in [t_n, t_{n+1}]} \left| \frac{d^2 f(c)}{dt^2} \right|$$

$$\begin{aligned} E &\leq \frac{e_r + e_r}{\Delta t} + \frac{\Delta t^2}{2} M \\ &= \frac{2e_r}{\Delta t} + \frac{M \Delta t^2}{2} \end{aligned}$$

As Δt gets large the error grows quadratically (ie. as Δt^2) due to the truncation error:

$$\frac{M\Delta t^2}{2}$$

As Δt gets small the error is dominated by the round-off error (ie. which grows as $\frac{1}{\Delta t}$) due to the round-off error:

$$\frac{2e_r}{\Delta t}$$

We are interested in minimizing the total error so we will choose a Δt in our implementations to minimize the error.

$$\frac{\partial |E|}{\partial (\Delta t)} = -\frac{2e_r}{\Delta t^2} + M\Delta t = 0$$

This implies that the Δt which minimizes the error should be:

$$\Delta t = \left(\frac{2e_r}{M} \right)^{1/3}$$

For a computer round-off error of $\sim 10^{-16}$ we would ideally like to use $\Delta t \sim 10^{-5}$ when using this first order forward difference formula.

Lets try coding this first order forward difference scheme in MATLAB on the function $\cos(t)$.

```

dt = 10(-1);    % Choose a good time step
t = (-1:dt:1);  % Define the domain
f = cos(t);     % Define the function
ft = 0*f;       % Initialize derivative ft
n = length(t);  % Calculate number of time steps

% Implement first order forward difference
% approximation of the first derivative
for j = 1:n-1,
    ft(j) = (f(j+1) - f(j))/dt;
end

% Or more efficiently
% ft(1:n-1) = (f(2:n) - f(1:n-1))/dt;

% Use a second order backward difference for the
% end point {Table 6. in Prof Kutz's notes}
ft(n) = (3*f(n)-4*f(n-1)+f(n-2))/(2*dt);

% Now plot f'(t) = -sin(t) and the numerical
% derivative to compare the results
plot(t,-sin(t),'r-',t,ft,'b*');

```

From the analysis carried through

- You can follow the same analysis for higher order approximations to the derivative.
- You can use not only forward difference schemes but also backward difference schemes, and centered difference schemes. See Table 6. of Kutz's notes for a table of $O(\Delta^2)$ forward and backward-difference schemes.
- You can use second order accurate approximations ($O(\Delta t^2)$, $O(\Delta t^3)$, $O(\Delta t^4)$, etc. See Table 4. and 5. of Kutz's notes for a table of some $O(\Delta t^2)$ and $O(\Delta t^2)$ centered-difference schemes.
- When implementing these methods be mindful of the end points (and the points near the end points for higher order difference schemes) since the end points don't have neighboring points outside of the data domain.