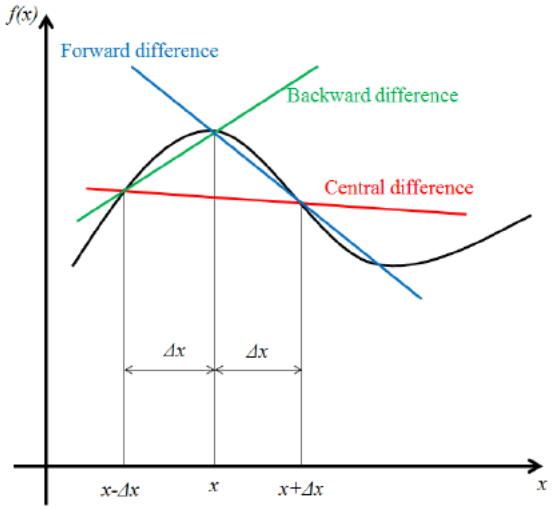
Differentiation:
The Finite Difference Method

MATH2070: Numerical Methods in Scientific Computing I

Location: http://people.sc.fsu.edu/~jburkardt/classes/math2070_2019/differentiation_difference/differentiation_difference.pd

---



Difference estimates for the derivative

Estimating Derivatives

Given the formula for a function $f(x)$:

- *how can we estimate derivatives, in particular, $f'(x)$?*
- *how does the finite difference stepsize affect the error?*
- *how does roundoff error limit the achievable accuracy?*

1 The forward difference estimate to $f'(x)$:

Suppose we have a formula, or a computational procedure, that provides us with the value $f(x)$ of a function, but now we need to estimate the derivative of the function, and calculus is not an option. The very definition of the derivative says it is the limit of the ratio of differences in function values to differences in argument values. Therefore, if h is "small enough", we can expect a good estimate for $f'(x)$ using the forward difference, symbolized by **fd(f,x,h)**:

$$f'(x) \approx \mathrm{fd}(f,x,h) \equiv \frac{f(x+h)-f(x)}{h}$$

It's easy to create a corresponding MATLAB function that will produce the forward difference estimate:

```
1 function value = fd ( f, x, h )
2
3   value = ( f ( x + h ) - f ( x ) ) / h;
```

```
4
5   return
6 end
```

Notice that the first argument is not a number, but the function. When using `fd()`, then, we can't simply pass a formula or function name; we have to identify this argument as a function, and specify its argument. Here are a few examples:

```
1   value = fd ( @(x)tan(x), x, h );
2   value = fd ( @(x)log(x+1), x, h );
3   value = fd ( @(x)x^2+3*x+4, x, h );    % A formula
```

Practice this command by setting `x=1` and `h=0.1` and then trying each of the three examples above. You should see answers of 4.0735, 0.4879, and 5.1.

Notice that `fd()` can also be called with a vector `x`. That means we can make plots comparing the exact and estimated derivatives. Try the following commands:

```
1   x = linspace ( 0.0, 2.0, 101 )   % Remember this command?
2   h = 0.1;
3   y = fd (@(x)log(x+1), x, h );
4   y2 = 1 ./ ( x + 1 );
5   plot ( x, y, 'r-', x, y2, 'b-' );  % Draw red and blue graphs together.
```

What "breaks" if you repeat this computation with the command:

```
1 value = fd ( @(x)x^2+3*x+4, x, h );
```

Can we fix the problem?

2 Discretization and roundoff errors

In exact arithmetic, the difference between the true and estimated values is known as the **discretization error**. We can estimate the order of the absolute value of the discretization error:

$$|f'(x) - \mathrm{fd}(f, x, h)| = O(h)$$

This formula seems to promise that we can drive the finite difference estimate towards the true derivative value simply by relentless decreasing h. But this is only half of the story!

As explained in class, as h is decreased, we began to make increasingly dangerous arithmetic errors. The quantities $f(x+h)$ and $f(x)$ become very close, so the number of significant digits in their difference decreases, and so we compute something slightly different from the exact arithmetic forward difference. This is known as **roundoff error**. As h gets smaller, the discretization error decreases, but the roundoff error rises. This means that at some point, we can expect our total error to begin to increase, and after that, our computed estimates are worthless.

3 Compare the forward difference estimate to the exact derivative

In class, the example of estimating the derivative of the tangent function at $x = \frac{\pi}{4}$ was used. We know that $tan'(\frac{\pi}{4}) = 2$. We imagine that decreasing h gives us better and better estimates. Let's see if this is true.

For convenience, let's modify our forward difference function so that it can compute the estimates for a vector of h values:

```
function value = fd ( f, x, h )

  value = ( f ( x + h ) - f ( x ) ) ./ h;

  return
end
```

It's a tiny difference, but it will make things much easier. Now our input `h` can be a list of steps of decreasing size, and we can compute all the finite difference estimates at once.

```
  h = 0.5;
  alpha = 0.5;
  n = 50
  p = 0 : n - 1;                % p = [ 0, 1, 2, ..., 49 ]
  h_list = h * alpha .^ p;      % h_list = [ h, h*alpha, h*alpha^2, ..., h*alpha^(n-1)
  fd_list = fd ( @(x)tan(x), pi/4, h_list )
  e_list = abs ( 2.0 - fd_list );
  plot ( h_list, e_list );
```

Looking at the plot, everthing seems great.

But if you actually look at the `fd_list` values, you can see that something odd is happening. At first the values seem to converge towards 2, but then they pull away again. (You may want to try the `format long` statement to see more digits in these values.) Normally, a plot reveals information, but this time, it seems to be hiding things from us. What is going wrong?

If you think about it, all the bad stuff is happening at extremely small values of h, so in order to see the disaster, we would need a microscope to be able to examine the lower left corner of the plot. Unfortunately, the MATLAB plotting algorithm doesn't actually plot these values because they are so close together!

When numbers vary greatly in scale, one way to try to fit them into a plot is to work with the logarithm.

```
  h = 0.25;
  alpha = 0.5;
  n = 50;
  p = 0 : n - 1;
  h_list = h * alpha .^ p;
  fd_list = fd ( @(x)tan(x), pi/4, h_list );
  e_list = abs ( 2.0 - fd_list );
  plot ( log ( h_list ), log ( abs ( e_list ) ) );
```

Now the information in our data can be displayed on the plot in a way that tells a story! We can use the `min()` command to get the value (1.5e-08) and the index (26) of the minimum entry in `e_list`.

```
  [ e_min, i_min ] = min ( e_list );
  fprintf ( '  Minimum error is at index %d\n', i_min );
  fprintf ( '  Minimum error is %g\n', e_min );
  fprintf ( '  Best h is %g\n', h_list(i_min) );
  fprintf ( '  Best estimate is %20.16g\n', fd_list(i_min) );
```

This means that our best estimate for the derivative was `fd_list(26)=2.000000014901161`. It means that the minimum useful value of `h` is `h_list(26)` or 7.45e-09, which is about half of the square root of the machine precision `sqrt(eps)=1.49e-08`.

4 Assignment #2

A better estimate for $f'(x)$ uses a centered difference. We can symbolize this by **center(f,x,h)**:

$$f'(x) \approx \text{center}(f,x,h) \equiv \frac{f(x+h) - f(x-h)}{2h}$$

We can estimate the order of the absolute value of the discretization error:

$$|f'(x) - \text{center}(f, x, h)| = O(h^2)$$

In class, it was suggested that this improved accuracy meant that the centered difference error would continue to decrease down to a smaller h than we saw for the forward difference.

Write a corresponding MATLAB function `center.m` to compute the centered difference estimate of the first derivative. Repeat the exercise for estimating the first derivative of $\tan(x)$ at $x = \frac{pi}{4}$, using 50 values of h, starting at $h = 0.25$ and dividing by half each time. Determine the value of h at which you made the smallest error, given that the correct derivative is 2.

Your program, which might be called *center_test.m*, should

1. set up 50 decreasing values of h in `h_list`;
2. use your `center()` function to compute the centered difference estimates `cd_list` for all the values in `h_list`;
3. determine the value of h in `h_list` at which you made the smallest error;
4. plot `h_list` versus `cd_list`;

Send me your program via email. I don't need a copy of your plot. The assignment is due by Friday evening, September 6.