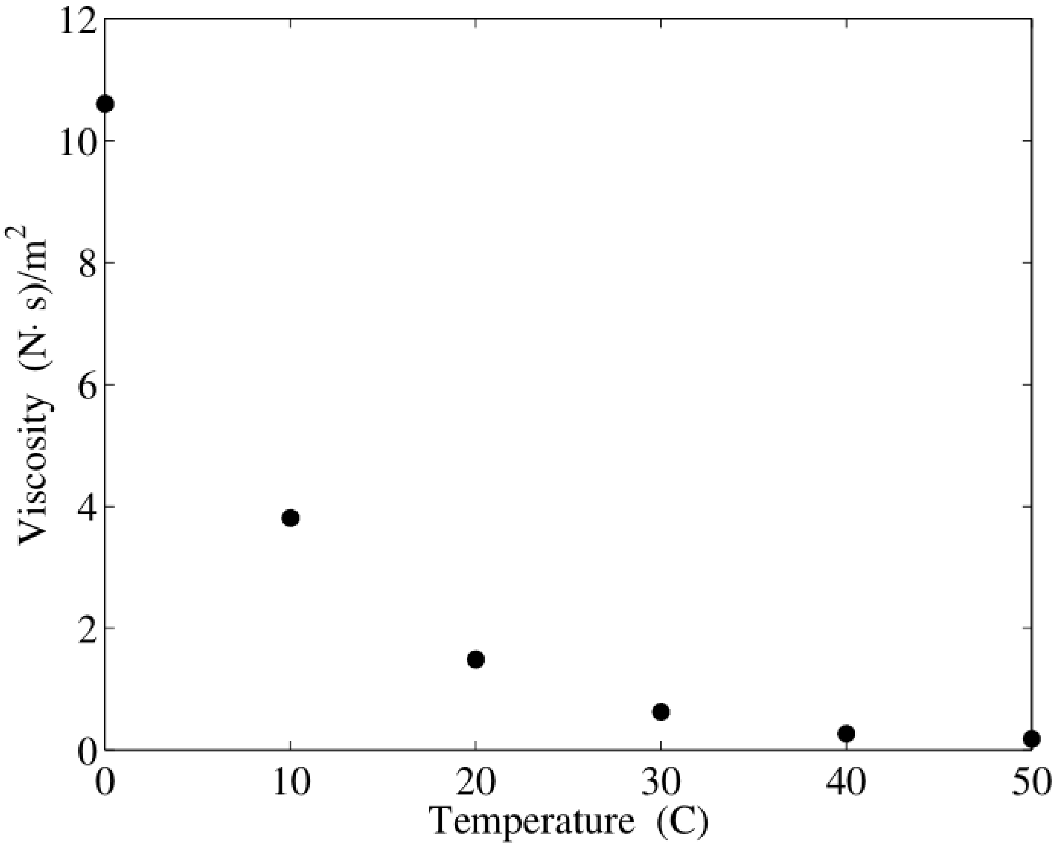
PHYS 351
Derivatives

Dr. Daugherity
Abilene Christian University

Problem



Estimate the derivative of a function from data points.

Related: if I get to calculate my own data points, what is the optimal spacing to use?

Approaches

Strategies:

1) Fit the data to a known function, take the analytic derivative
2) Use cubic spline interpolation, take derivative of splines
3) Calculate the result "directly"

Errors

Two types of ultimately unavoidable errors in computing:

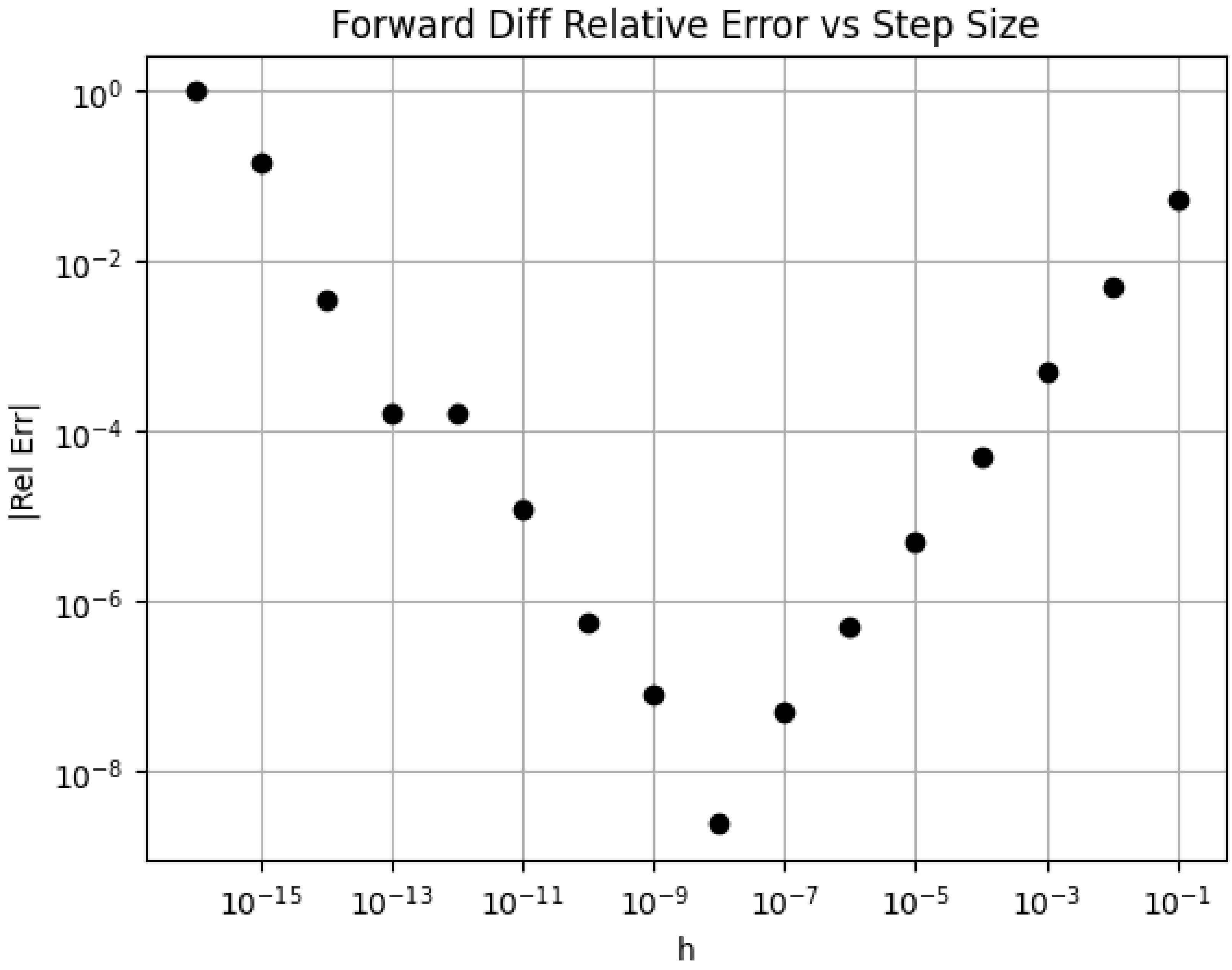
1) Round off
2) Truncation

Derivation

1) Forward Difference

Challenge

Pick an easy-to-differentiate function $f(x)$ and an x_0 value to estimate $f'(x_0)$. What value of h gives you the smallest error?



f(x) = sin x
at x=1

Using forward difference

Derivation

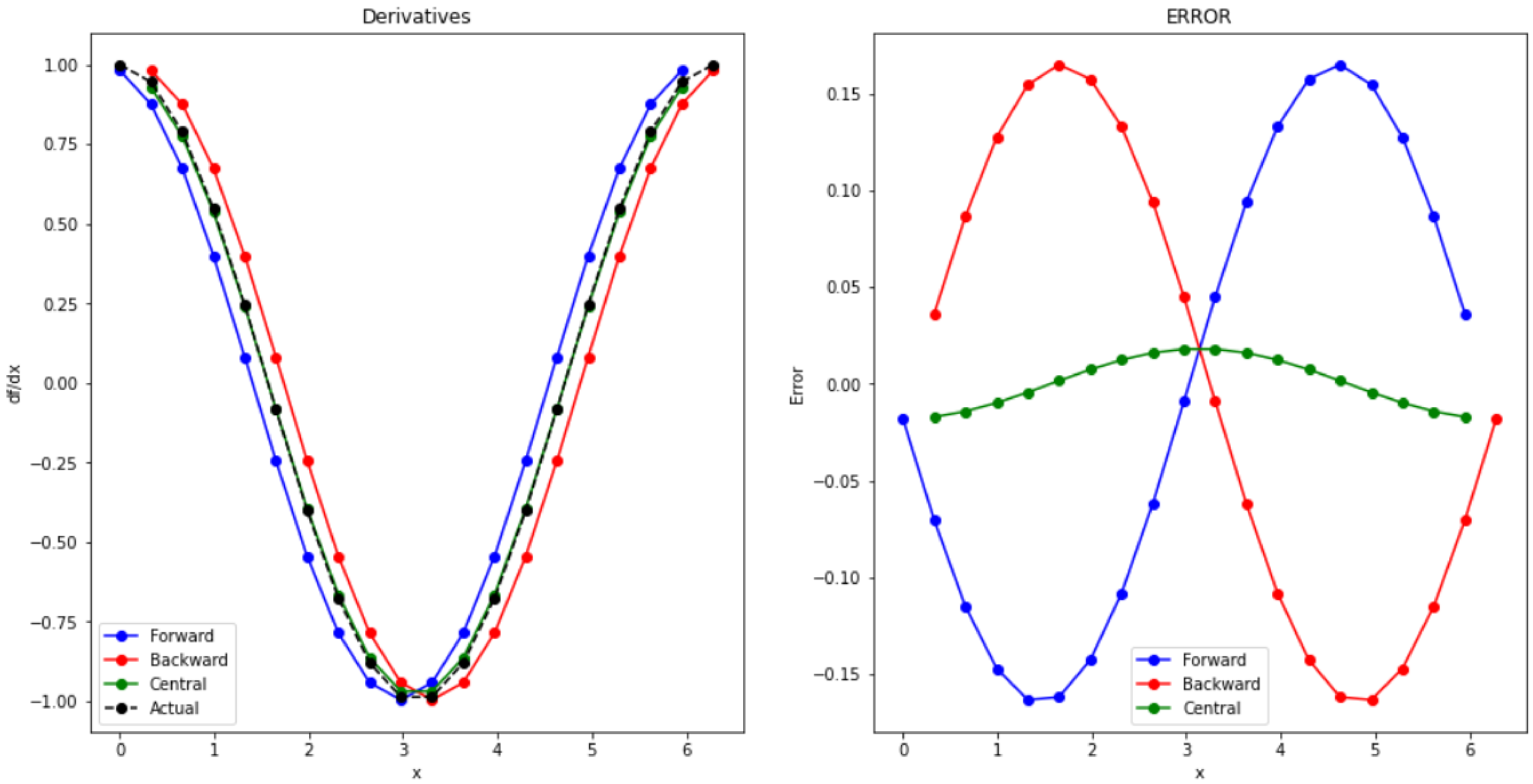
1) Forward Difference
2) Backward Difference
3) Centered Difference

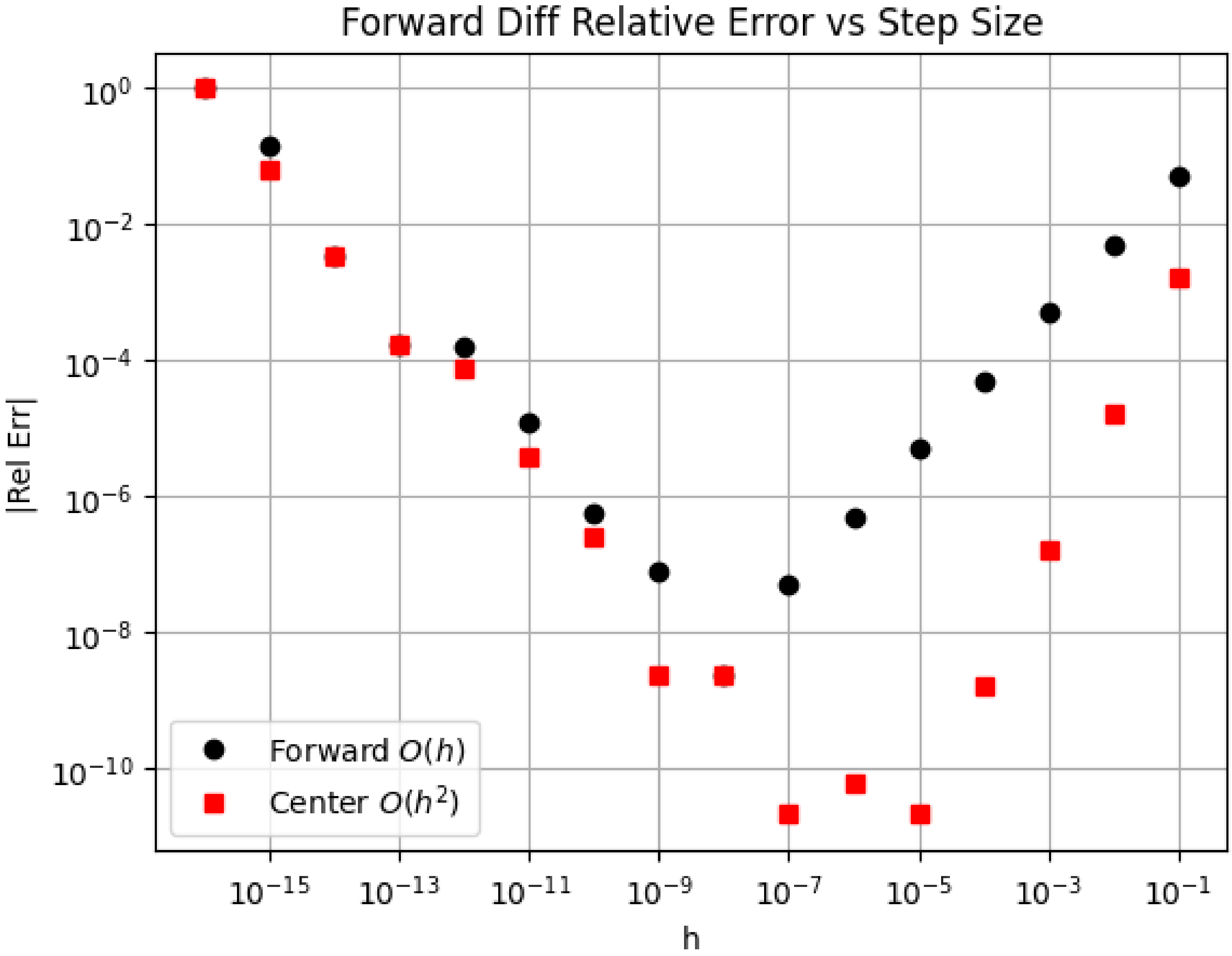
Challenge

1) Check if centered difference really is more accurate

2) Now what value of h gives you the smallest error?

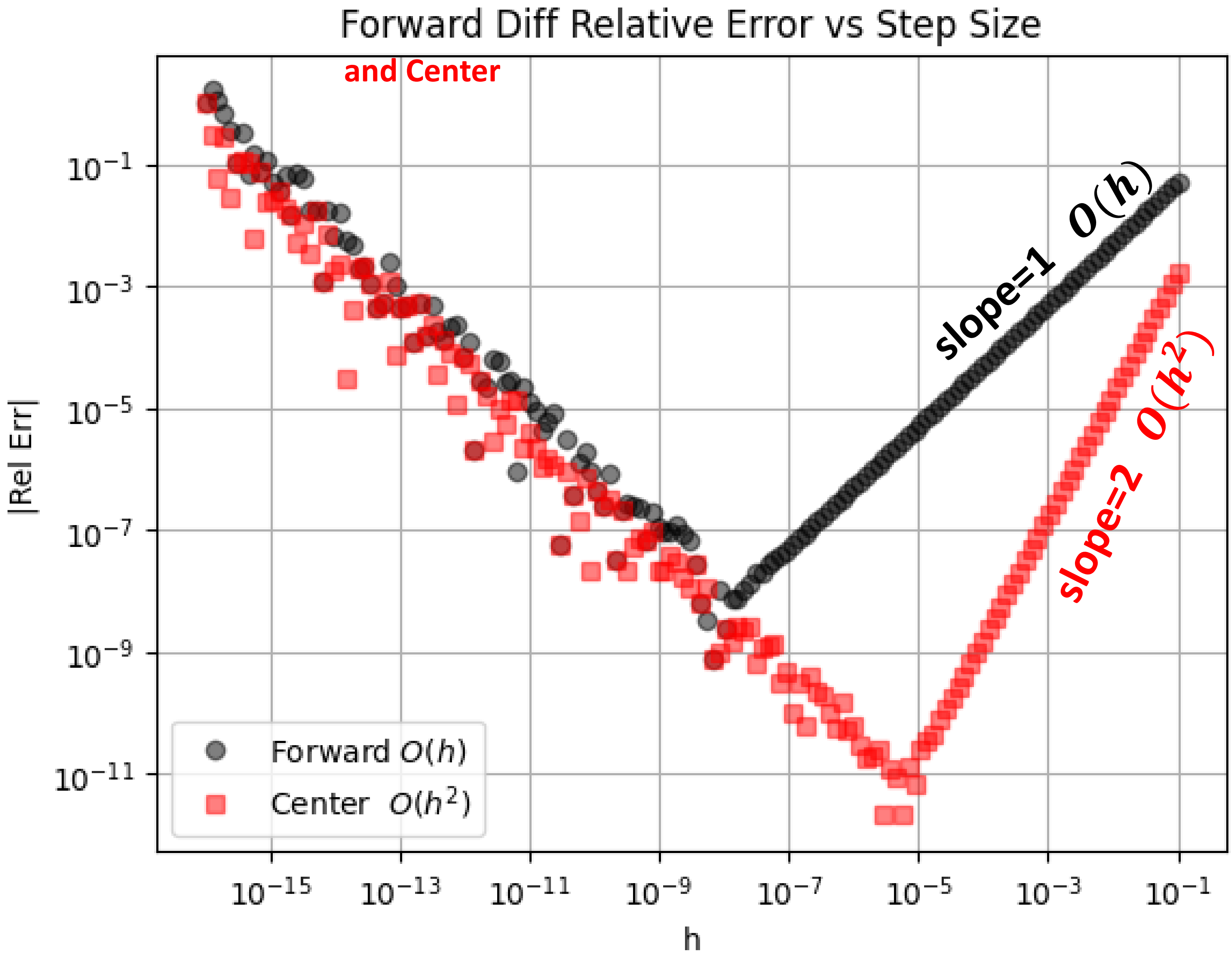
Taylor Series Methods





Forward Diff Relative Error vs Step Size
|Rel Err|
h
Forward $O(h)$
Center $O(h^2)$
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
10^{-15}
10^{-13}
10^{-11}
10^{-9}
10^{-7}
10^{-5}
10^{-3}
10^{-1}

f(x) = sin x
at x=1



Region on the left (small h) the dominant error is ROUNDING from subtractive cancellation in the numerator.

Region on the right (larger h) the dominant error is TRUNCATION from only keeping lower order approximations. The slopes show the order of the error!

Continuous Function

https://docs.scipy.org/doc/scipy/reference/generated/scipy.misc.derivative.html#scipy.misc.derivative

Miscellaneous routines (`scipy.misc`)

Deprecated since version 1.10.0: This module is deprecated and will be completely removed in SciPy v2.0.0.

Various utilities that don’t have another home.

`ascent` ()	Get an 8-bit grayscale bit-depth, 512 x 512 derived image for easy use in demos
`central_diff_weights` (Np[, ndiv])	Return weights for an Np-point central derivative.
`derivative` (func, x0[, dx, n, args, order])	Find the nth derivative of a function at a point.
`face` ([gray])	Get a 1024 x 768, color image of a raccoon face.
`electrocardiogram` ()	Load an electrocardiogram as an example for a 1-D signal.

scipy.misc.**derivative**(*func, x0, dx=1.0, n=1, args=(), order=3*)

Find the nth derivative of a function at a point.

Given a function, use a central difference formula with spacing *dx* to compute the nth derivative at *x0*.

Parameters:
- func : *function*
  Input function.
- x0 : *float*
  The point at which the nth derivative is found.
- dx : *float, optional*
  Spacing.
- n : *int, optional*
  Order of the derivative. Default is 1.
- args : *tuple, optional*
  Arguments
- order : *int, optional*
  Number of points to use, must be odd.

Notes

Decreasing the step size too small can result in round-off error.

Use `scipy.misc.derivative` for centered-difference

Be careful with dx!!! Both too big and too small are bad. Something in the middle is best like `1e-5`

Data Points

Easy! Use numpy.gradient. Lots of choices for 2nd parameter to specify dx

numpy.gradient(*f*, **varargs*, *axis=None*, *edge_order=1*) [source]

Return the gradient of an N-dimensional array.

The gradient is computed using second order accurate central differences in the interior points and either first or second order accurate one-sides (forward or backwards) differences at the boundaries. The returned gradient hence has the same shape as the input array.

Parameters: **f** : *array_like*

An N-dimensional array containing samples of a scalar function.

varargs : *list of scalar or array, optional*

Spacing between f values. Default unitary spacing for all dimensions. Spacing can be specified using:

1. single scalar to specify a sample distance for all dimensions.
2. N scalars to specify a constant sample distance for each dimension. i.e. *dx*, *dy*, *dz*, ...
3. N arrays to specify the coordinates of the values along each dimension of F. The length of the array must match the size of the corresponding dimension
4. Any combination of N scalars/arrays with the meaning of 2. and 3.

https://numpy.org/doc/stable/reference/generated/numpy.gradient.html

Higher Order

https://en.wikipedia.org/wiki/Finite_difference_coefficient

Derivative	Accuracy	−5	−4	−3	−2	−1	0	1	2	3	4	5
1	2					−1/2	0	1/2				
	4				1/12	−2/3	0	2/3	−1/12			
	6			−1/60	3/20	−3/4	0	3/4	−3/20	1/60		
	8		1/280	−4/105	1/5	−4/5	0	4/5	−1/5	4/105	−1/280	
2	2					1	−2	1				
	4				−1/12	4/3	−5/2	4/3	−1/12			
	6			1/90	−3/20	3/2	−49/18	3/2	−3/20	1/90		
	8		−1/560	8/315	−1/5	8/5	−205/72	8/5	−1/5	8/315	−1/560	
3	2				−1/2	1	0	−1	1/2			
	4			1/8	−1	13/8	0	−13/8	1	−1/8		
	6		−7/240	3/10	−169/120	61/30	0	−61/30	169/120	−3/10	7/240	
4	2				1	−4	6	−4	1			
	4			−1/6	2	−13/2	28/3	−13/2	2	−1/6		
	6		7/240	−2/5	169/60	−122/15	91/8	−122/15	169/60	−2/5	7/240	
5	2			−1/2	2	−5/2	0	5/2	−2	1/2		
	4		1/6	−3/2	13/3	−29/6	0	29/6	−13/3	3/2	−1/6	
	6	−13/288	19/36	−87/32	13/2	−323/48	0	323/48	−13/2	87/32	−19/36	13/288

For example, the third derivative with a second-order accuracy is

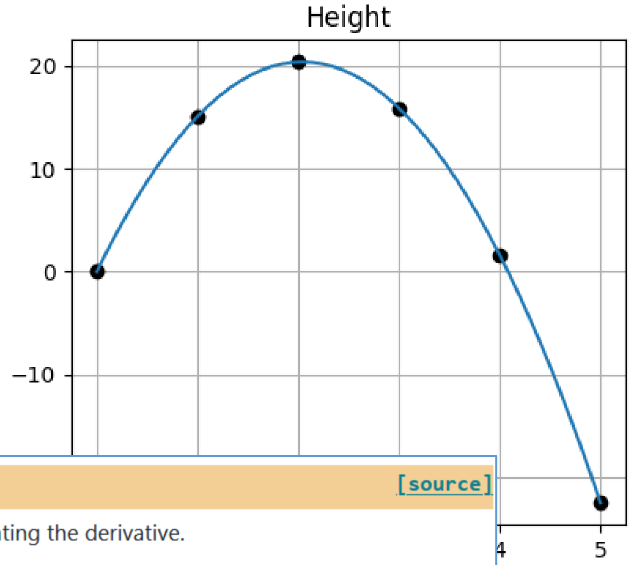
$$f'''(x_0) \approx \frac{-\frac{1}{2}f(x_{-2}) + f(x_{-1}) - f(x_{+1}) + \frac{1}{2}f(x_{+2})}{h_x^3} + O\left(h_x^2\right)$$

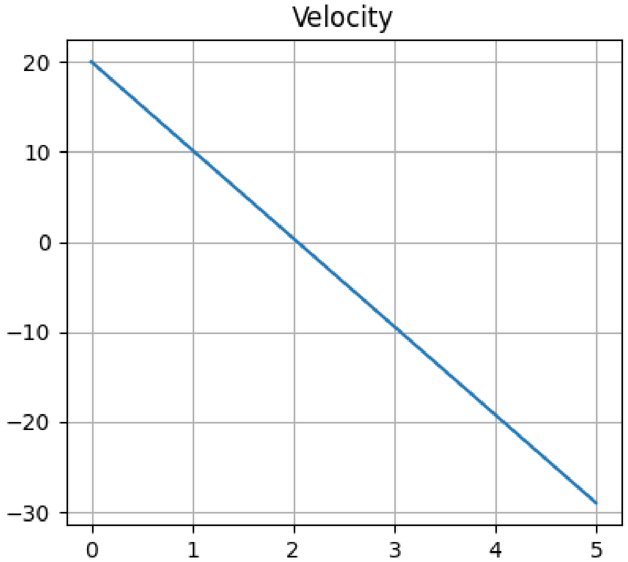
https://web.media.mit.edu/~crtaylor/calculator.html - nice tool which gives python code!

Cubic Spline Interpolation

https://github.com/mdaugherity/Numerical2024/blob/main/fits/Week_6_Interpolation.ipynb

```python
# Make fake data for object in free-fall
tdata = np.arange(0,6)
v0 = 20
g = 9.8
ydata = v0*tdata - 0.5*g*tdata**2

# Fit cubic spline
yspline = CubicSpline(tdata, ydata)
vspline = yspline.derivative()  # automatically get derivative

ts = np.linspace(0,5)

plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
plt.plot(tdata, ydata, 'ko')
plt.plot(ts,yspline(ts))
plt.title('Height')
plt.grid()

plt.subplot(1,2,2)
plt.plot(ts,vspline(ts))
plt.title('Velocity')
plt.grid()
```





derivative(*nu=1*) [source]

Construct a new piecewise polynomial representing the derivative.

Parameters:

nu : ***int, optional***

Order of derivative to evaluate. Default is 1, i.e., compute the first derivative. If negative, the antiderivative is returned.

Returns:

pp : ***PPoly***

Piecewise polynomial of order k2 = k - n representing the derivative of this polynomial.

Notes

Derivatives are evaluated piecewise for each polynomial segment, even if the polynomial is not differentiable at the breakpoints. The polynomial intervals are considered half-open, `[a, b)`, except for the last interval which is closed `[a, b]`.

https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.CubicSpline.derivative.html#scipy.interpolate.CubicSpline.derivative

DrD’s Advice

- Data points (that you can’t change):
  - use numpy.gradient for centered difference or spline interpolation
  - Noisy data? You should fit it first and differentiate the fit
- Function (that you can sample):
  - sample it with linspace and use numpy.gradient (remember that error goes as h^2)
  - or to get the derivative at a single point just write your own centered difference and use h=1e-5 or so
  - (but you probably don’t want to sample the entire function with that step size because that is millions of points…)
- If I really had a situation where this wasn’t good enough I would just use another library instead of looking up and implementing higher-order stuff myself:
  - https://github.com/maroba/findiff
  - https://github.com/pbrod/numdifftools

Things to Know

- Roundoff vs Truncation Errors
- CENTERED DIFFERENCE IS BEST

  most accurate step size is about h=1e-5

- Use `numpy.gradient` for data points
- Fitting / Spline Interpolation is also an option