Machine Learning and Photonics

Math Foundations: Numerical Differentiation

Ergun Simsek

University of Maryland Baltimore County

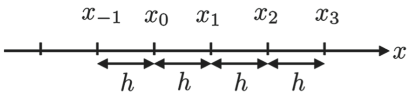
simsek@umbc.edu

If $f(t, x)$ exists, you can determine speed and acceleration!

February 7, 2023

Numerical Grid

1. **Numerical grid**: An evenly spaced set of points over the domain of a function (i.e., the independent variable), over some interval.
2. The **spacing** or **step size** of a numerical grid: The distance between adjacent points on the grid.
3. If x is a numerical grid, then x_j is the j^{th} point in the numerical grid and h is the spacing between x_{j-1} and x_j.



Numerical Grid: 1D vs 2D

Both in MATLAB and Python, you can easily create a numerical grid using linspace

linspace(0, 2, 5) ==> [0 0.5000 1.0000 1.5000 2.0000]

However, be careful

logspace(0, 2, 5) ==> [1.0000 3.1623 10.0000 31.6228 100.0000]

2D Grid

You can use *meshgrid* to create a 2D mesh. TRY IT!

Question: Can you drive an analytical expression for logspace(a, b, N)?

Finite Difference Approximating Derivatives

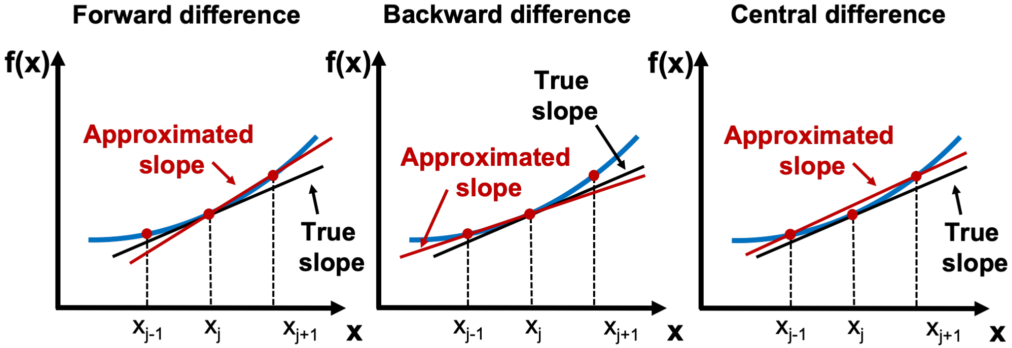
The derivative $f'(x)$ of a function $f(x)$ at the point $x = a$ is defined as:

$$f'(a) = \lim_{x \to a} \frac{f(x) - f(a)}{x - a}$$

- The derivative at $x = a$ is the slope at this point.
- In **finite difference** approximations of this slope, we can use values of the function in the neighborhood of the point $x = a$ to achieve the goal.

Finite Difference Approximating Derivatives (Cont...)

To estimate the slope of the function at x_j



$$f'(x_j) = \frac{f(x_{j+1}) - f(x_j)}{x_{j+1} - x_j}$$

$$f'(x_j) = \frac{f(x_j) - f(x_{j-1})}{x_j - x_{j-1}}$$

$$f'(x_j) = \frac{f(x_{j+1}) - f(x_{j-1})}{x_{j+1} - x_{j-1}}$$

Finite Difference Approximating Derivatives with Taylor Series

For an arbitrary function $f(x)$ the Taylor series of f around $a = x_j$ is

$$f(x) = \frac{f(x_j)(x - x_j)^0}{0!} + \frac{f'(x_j)(x - x_j)^1}{1!} + \frac{f''(x_j)(x - x_j)^2}{2!} + \frac{f'''(x_j)(x - x_j)^3}{3!} + \cdots .$$

If x is on a grid of points with spacing h, we can compute the Taylor series at $x = x_{j+1}$ to get

$$f(x_{j+1}) = \frac{f(x_j)(x_{j+1} - x_j)^0}{0!} + \frac{f'(x_j)(x_{j+1} - x_j)^1}{1!} + \frac{f''(x_j)(x_{j+1} - x_j)^2}{2!} + \frac{f'''(x_j)(x_{j+1} - x_j)^3}{3!} + \cdots$$

Finite Difference Approximating Derivatives with Taylor Series

Substituting $h = x_{j+1} - x_j$ and solving for $f'(x_j)$ gives the equation

$$f'(x_j) = \frac{f(x_{j+1}) - f(x_j)}{h} + \left(-\frac{f''(x_j)h}{2!} - \frac{f'''(x_j)h^2}{3!} - \cdots \right).$$

The terms that are in parentheses are called **higher order terms** of h, which can be rewritten as

$$-\frac{f''(x_j)h}{2!} - \frac{f'''(x_j)h^2}{3!} - \cdots = h(\alpha + \epsilon(h)),$$

where α is some constant, and $\epsilon(h)$ is a function of h that goes to zero as h goes to 0.

Finite Difference Approximating Derivatives with Taylor Series

We use the abbreviation "$\mathcal{O}(h)$" for $h(\alpha + \epsilon(h))$, and in general, we use the abbreviation "$\mathcal{O}(h^p)$" to denote $h^p(\alpha + \epsilon(h))$.

Substituting $\mathcal{O}(h)$ into the previous equations gives

$$f'(x_j) = \frac{f(x_{j+1}) - f(x_j)}{h} + \mathcal{O}(h).$$

This gives the **forward difference** formula for approximating derivatives as

$$f'(x_j) \approx \frac{f(x_{j+1}) - f(x_j)}{h},$$

and we say this formula is $\mathcal{O}(h)$.

By computing the Taylor series around $a = x_j$ at $x = x_{j-1}$ and again solving for $f'(x_j)$, you can get the **backward difference** formula

Central Difference Approximation with Taylor Series

First, let's compute the Taylor series around $a = x_j$ at both x_{j+1} and x_{j-1}

$$f(x_{j+1}) = f(x_j) + f'(x_j)h + \frac{1}{2}f''(x_j)h^2 + \frac{1}{6}f'''(x_j)h^3 + \cdots$$

and

$$f(x_{j-1}) = f(x_j) - f'(x_j)h + \frac{1}{2}f''(x_j)h^2 - \frac{1}{6}f'''(x_j)h^3 + \cdots .$$

Then let's subtract them

$$f(x_{j+1}) - f(x_{j-1}) = 2f'(x_j)h + \frac{2}{3}f'''(x_j)h^3 + \cdots ,$$

Finally, solve for $f'(x_j)$ (Note: this time it's $\mathcal{O}(h^2)$)==>

$$f'(x_j) \approx \frac{f(x_{j+1}) - f(x_{j-1})}{2h}.$$

More Points ==> Higher Accuracy

Let's take the Taylor series of f around $a = x_j$ and compute the series at $x = x_{j-2}, x_{j-1}, x_{j+1}, x_{j+2}$

$$
\begin{aligned}
f(x_{j-2}) &= f(x_j) - 2hf'(x_j) + \frac{4h^2 f''(x_j)}{2} - \frac{8h^3 f'''(x_j)}{6} + \frac{16h^4 f''''(x_j)}{24} - \frac{32h^5 f'''''(x_j)}{120} + \cdots \\
f(x_{j-1}) &= f(x_j) - hf'(x_j) + \frac{h^2 f''(x_j)}{2} - \frac{h^3 f'''(x_j)}{6} + \frac{h^4 f''''(x_j)}{24} - \frac{h^5 f'''''(x_j)}{120} + \cdots \\
f(x_{j+1}) &= f(x_j) + hf'(x_j) + \frac{h^2 f''(x_j)}{2} + \frac{h^3 f'''(x_j)}{6} + \frac{h^4 f''''(x_j)}{24} + \frac{h^5 f'''''(x_j)}{120} + \cdots \\
f(x_{j+2}) &= f(x_j) + 2hf'(x_j) + \frac{4h^2 f''(x_j)}{2} + \frac{8h^3 f'''(x_j)}{6} + \frac{16h^4 f''''(x_j)}{24} + \frac{32h^5 f'''''(x_j)}{120} + \cdots
\end{aligned}
$$

More Points ==> Higher Accuracy

To get the h^2, h^3, and h^4 terms to cancel out, we can compute

$$f(x_{j-2}) - 8f(x_{j-1}) + 8f(x_{j-1}) - f(x_{j+2}) = 12hf'(x_j) - \frac{48h^5 f'''''(x_j)}{120}$$

which can be rearranged to

$$f'(x_j) = \frac{f(x_{j-2}) - 8f(x_{j-1}) + 8f(x_{j-1}) - f(x_{j+2})}{12h} + O(h^4).$$

Some Tips

Both in MATLAB and Python, you can compute finite differences directly.
For a vector f, the command $d = diff(f)$; and $d = np.diff(f)$, respectively, produces an array d in which the entries are the differences of the adjacent elements in the initial array f.
i.e., $d(i) = f(i+1) - f(i)$.

The length of d will be $N-1$, assuming f has N data points.

Example: $f(x) = \cos(x)$

```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('seaborn-poster')
%matplotlib inline
```

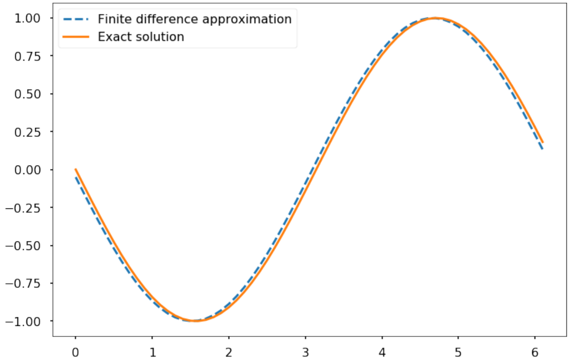
Note that *matplotlib inline* allows us to have the plots directly below the cell in the notebook.

```
# step size
h = 0.1
# define grid
x = np.arange(0, 2*np.pi, h)
# compute function
y = np.cos(x)

# compute vector of forward differences
forward_diff = np.diff(y)/h
# compute corresponding grid
x_diff = x[:-1:]
# compute exact solution
exact_solution = -np.sin(x_diff)
```

Example: $f(x) = \cos(x)$ (Cont...)

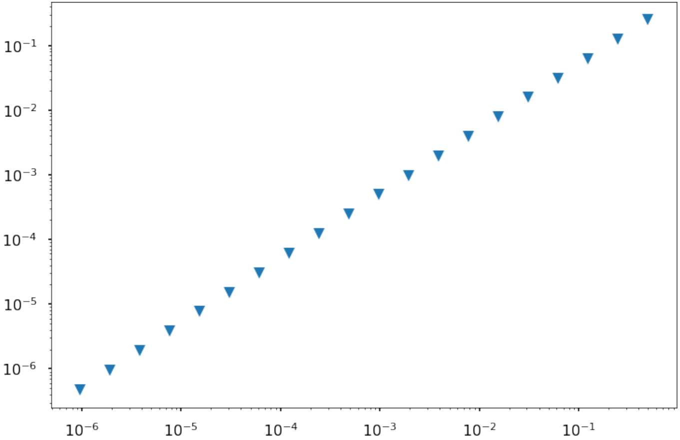
```python
# Plot solution
plt.figure(figsize = (12, 8))
plt.plot(x_diff, forward_diff, '--', \
         label = 'Finite difference approximation')
plt.plot(x_diff, exact_solution, \
         label = 'Exact solution')
plt.legend()
plt.show()

# Compute max error between
# numerical derivative and exact solution
max_error = max(abs(exact_solution - forward_diff))
print(max_error)
```

Example: $f(x) = \cos(x)$ (Cont...)



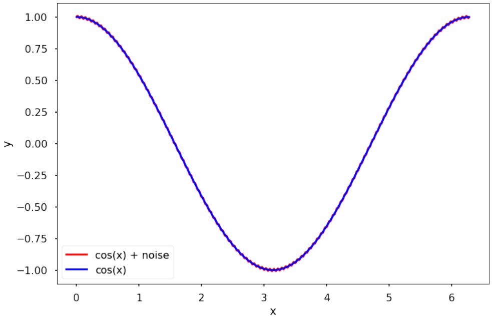
Maximum Error: 0.04999

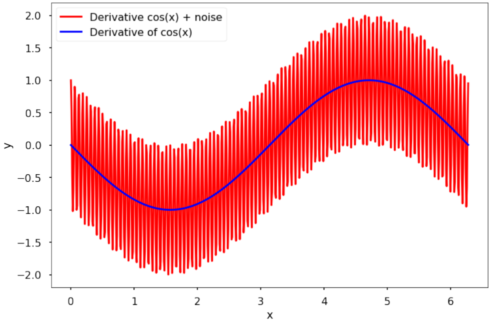
Step-size vs. Maximum Error



Numerical Differentiation with Noise

Consider $f(x) = \cos(x)$ vs. $f_{\epsilon,\omega}(x) = \cos(x) + \epsilon \sin(\omega x)$ where $0 < \epsilon \ll 1$ is a very small number and ω is a large number, e.g. $\epsilon = 0.01$ and $\omega = 100$





Approximating of Higher Order Derivatives

Let's take the Taylor series around $a = x_j$ and then compute it at $x = x_{j-1}$ and x_{j+1}

$$f(x_{j-1}) = f(x_j) - hf'(x_j) + \frac{h^2 f''(x_j)}{2} - \frac{h^3 f'''(x_j)}{6} + \cdots$$

and

$$f(x_{j+1}) = f(x_j) + hf'(x_j) + \frac{h^2 f''(x_j)}{2} + \frac{h^3 f'''(x_j)}{6} + \cdots .$$

If we add these two equations together, we get

$$f(x_{j-1}) + f(x_{j+1}) = 2f(x_j) + h^2 f''(x_j) + \frac{h^4 f''''(x_j)}{24} + \cdots ,$$

and with some rearrangement gives the approximation

$$f''(x_j) \approx \frac{f(x_{j+1}) - 2f(x_j) + f(x_{j-1})}{h^2},$$

and is $\mathcal{O}(h^2)$.