Numerical Differentiation

2.1 Accuracy of Differentiation

Consider a simple periodic function

$$f(x) = \sin x \tag{2.1}$$

on $x \in [0, 2\pi]$. The analytical derivative is $f'(x) = \cos x$.

On a grid with N data points on $[0, 2\pi)$ we will have $\Delta x = 2\pi/N$. On such a grid, we could evaluate the derivative as

$$f'(x) \approx \frac{f(x + \Delta x) - f(x)}{\Delta x} \tag{2.2}$$

(a) Using $N = 20$ gridpoints (`np.linspace(0, 2 * np.pi, 20, endpoint=False)` evaluate the numerical derivative of $f(x)$ using the above formula. Make sure you account for periodicity. Plot the result against the true $f'(x)$.
Hint: `np.roll` might be a useful function.

For smooth functions, we can do better than the above formula.

(b) Make the same plots for the following numerical derivative schemes:

Finite Difference Coefficients for the First Derivative

	$-3\Delta x$	$-2\Delta x$	$-\Delta x$	0	Δx	$2\Delta x$	$3\Delta x$
$\mathcal{O}(\Delta x)$	0	0	0	-1	1	0	0
$\mathcal{O}(\Delta x^2)$	0	0	$-\frac{1}{2}$	0	$\frac{1}{2}$	0	0
$\mathcal{O}(\Delta x^4)$	0	$\frac{1}{12}$	$-\frac{2}{3}$	0	$\frac{2}{3}$	$-\frac{1}{12}$	0
$\mathcal{O}(\Delta x^6)$	$-\frac{1}{60}$	$\frac{3}{20}$	$-\frac{3}{4}$	0	$\frac{3}{4}$	$-\frac{3}{20}$	$\frac{1}{60}$

(c) Evaluate the maximum absolute error made by the methods above over the entire domain $[0, 2\pi)$.

(d) Make a log-log plot of the maximum absolute error of each method as a function of $N \in [10, 10^6]$ (use e.g. `N = np.logspace(1, 6, 50, dtype=int)`) and explain the plot.

(e) What is the best accuracy that you obtain with the second order method? And with the sixth order method? Can you predict the best accuracy for the first order method without evaluating higher N?

(f) How many grid points are needed for a second order method to obtain the same accuracy as a fourth order method with $N = 100$ points?

(g) Compare the curves to a plot of $\sim \Delta x^n$, where n is the order of the method as stated in the above table. Also plot $5 \cdot 10^{-16}/\Delta x$ and compare it to the best error of the methods[1].

[1] $\epsilon \approx 5 \cdot 10^{-16}$ is the machine epsilon if you do calculations using double precision (which is `numpy` default).

2.2 Deriving Schemes

In this exercise you will calculate your own finite difference coefficients.

(a) Use `scipy.linalg.solve` along with the finite difference coefficient formulas of the main text to derive the coefficients of the table in the previous exercise.

At boundaries we cannot use central derivatives.

(b) Derive numerical schemes for the first and second derivatives at x which uses $f(x)$, $f(x+\Delta x)$, $f(x+2\Delta x)$ and $f(x+3\Delta x)$. What are the order of the methods derived?

Now consider the matrix

$$\mathbf{A} = \frac{1}{\Delta x}\begin{pmatrix} -1 & 1 & 0 & 0 & 0 \\ -0.5 & 0 & 0.5 & 0 & 0 \\ 0 & -0.5 & 0 & 0.5 & 0 \\ 0 & 0 & -0.5 & 0 & 0.5 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix} \tag{2.3}$$

When matrix-multiplied onto a vector this will take the first derivative: $\boldsymbol{f}' \approx \mathbf{A}\boldsymbol{f}$. This matrix uses second order method for all inner points, but only first order methods at the edges.

(c) *(optional)* Build a 10×10 matrix $\mathbf{A}$ such that matrix multiplication $\mathbf{A}\boldsymbol{f}$ calculates the second derivative of $\boldsymbol{f}$ using a second order method everywhere. Use central difference schemes wherever possible, but forward/backward schemes at the edges.
Note: The central scheme $(1, -2, 1)$ is second order despite only using

three stencil points. The forward scheme for the second derivative needs four points to be second order (as derived in (**b**)).

(d) *(optional)* Update your matrix to use a fourth order method at all points.

There is nothing preventing you from having a varying Δx. This is called using an *irregular grid.*

(e) *(optional)* Derive a scheme to make the best possible approximation of the second derivative of a function at $x = 0.5$, where you only have the function values $f(0.0)$, $f(0.1)$, $f(0.25)$, $f(0.6)$, and $f(1.0)$.

2.3 Discontinuities

Consider the following function

$$f(x) = \begin{cases} e^{-x} + ax - 1 & x < 0 \\ x^2 & x > 0 \end{cases} \tag{2.4}$$

which has the derivative

$$f'(x) = \begin{cases} -e^{-x} + a & x < 0 \\ 2x & x > 0 \end{cases} \tag{2.5}$$

(a) Plot the function and its derivative for $a = 0, a = 1, a = 2$.

(b) For what values of $a \in \mathbb{R}$ is $f(0)$ well-defined? i.e. for what values of a do $\lim_{x\to 0^-} f(x) = \lim_{x\to 0^+} f(x)$? What about $f'(0)$?

To numerically avoid the troublesome point $x = 0$, we can choose a grid-spacing that does not include this. This can e.g. be achieved by `x = np.linspace(-1, 1, N)` if N is even. Choose e.g. $N = 1000$.

(c) Using the *central* second order finite difference scheme [$\mathcal{O}(\Delta x^2)$], calculate the numerical derivative of $f(x)$ and compare to the analytical $f'(x)$ for $a = 0, a = 1, a = 2$. Plot your results near $x = 0$ and discuss for what values of a you find a good approximation.

(d) Derive a suitable finite difference scheme for calculating the first derivative of the function in such a way that points from $x < 0$ are never used together with points $x > 0$.

(e) Plot your results near $x = 0$ and compare with the central scheme.

Ordinary Differential Equations

3.1 Explicit vs. Implicit

Consider the ODE

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \alpha\left(\sin t - x\right). \tag{3.1}$$

For $x(0) = 0$, this has the analytical solution

$$x(t) = \frac{\alpha}{1+\alpha^2}\left(e^{-\alpha t} - \cos t + \alpha \sin t\right). \tag{3.2}$$

(a) Solve the equation using the explicit Euler method and compare with the analytical solution for $\alpha = 0.1$ on $t \in [0, 100]$ using $\Delta t = 0.01$.

(b) Do the same for the implicit Euler method.

(c) Investigate the behaviour of the two methods for $\alpha > 200$.

3.2 Van der Pol oscillator

The equation

$$\frac{\mathrm{d}^2x}{\mathrm{d}t^2} - \mu(1-x^2)\frac{\mathrm{d}x}{\mathrm{d}t} + x = 0 \tag{3.3}$$

is called the Van der Pol oscillator.

(a) Rewrite the equation to a system of two coupled first-order ODEs.

(b) Solve the equation using the Euler method for $\mu = 0.1$, $\mu = 1.0$, $\mu = 10$, $\mu = 100$, and $\mu = 250$ with initial conditions $x(0) = 2$ and $x'(0) = 0$. Solve for $t \in [0, 10\mu]$ and use $\Delta t = 0.01$.

(c) Solve the equation using SCIPY's `solve_ivp` with `method='BDF'` (this is a higher-order implicit method, "backwards differentiation formula").

(d) Plot the Euler and the BDF solution in the same plot for each value of μ.

As μ increases, the equations becomes more and more stiff. Thus implicit methods (such as BDF) become important.

(e) Solve the equation using `solve_ivp`'s `method='RK45'` and report the run time compared to that of BDF for each value of μ.

3.3 Runge–Kutta method

Independent exercise

The main text gives the following formula for the fourth order Runge–

Kutta method

$$
\begin{aligned}
k_1 &= F(f(t), t) \\
k_2 &= F(f(t) + \frac{1}{2}k_1\Delta t, t + \frac{1}{2}\Delta t) \\
k_3 &= F(f(t) + \frac{1}{2}k_2\Delta t, t + \frac{1}{2}\Delta t) \\
k_4 &= F(f(t) + k_3\Delta t, t + \Delta t) \\
f(t + \Delta t) &= f(t) + \frac{1}{6}\left[k_1 + 2k_2 + 2k_3 + k_4\right]\Delta t
\end{aligned}
\tag{3.4}
$$

(a) Implement this method to solve

$$f'(t) = 1 + \sin(t)f(t) \tag{3.5}$$

for $t \in [0, 15]$ using $\Delta t = 0.001$ and $f(0) = 0$. Also implement the Euler method for this ODE and check that it gives the same result.

(b) Now take $\Delta t = 1.0$ and solve the equation using the Runge–Kutta method. Plot the solution on top of the result with $\Delta t = 0.001$.

(c) Take $\Delta t = 0.25$ and solve the equation using the Euler method. Plot the solution on top of the result with $\Delta t = 0.001$ and comment on the result.

3.4 Dormand–Prince method*

In this exercise you are going to write your own version of the infamous 'ode45' method (aka 'RK45'). Upon completion you should feel enormously more confident using library version of this method as you will