

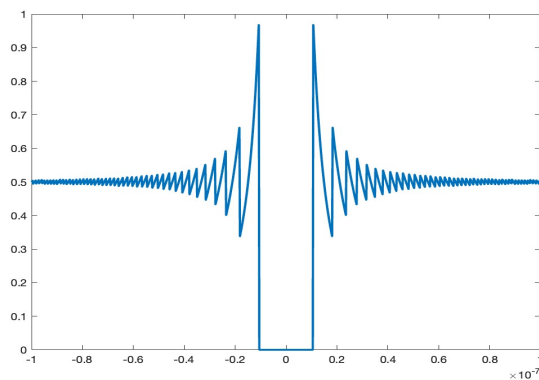
Numerical Analysis I

1. Create a function to compute $f(x) = (1 - \cos(x))/x^2$.
 - a. Create an array of evenly-spaced numbers in the interval $[-1, 1]$.
 - b. Apply f to the array of numbers and plot the result.
 - c. What happens when $x \approx 0$?

The intent of this exercise was to demonstrate the numerical instability of division a/b when $a \approx b$. Before writing any code, we *expect* $f(x)$ to remain numerically stable as x approaches 0; the below code snippets conduct our experiment.

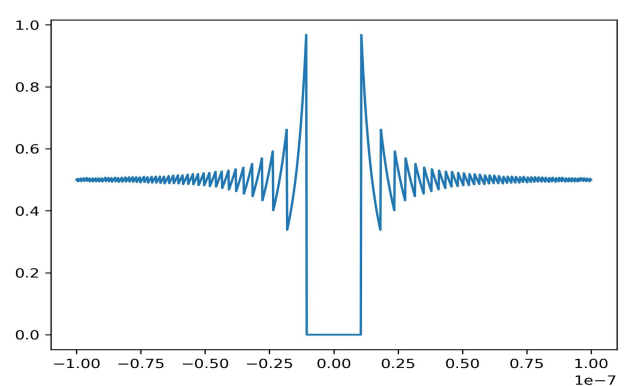
approximations.m

```
1 function y = f(x)
2     % Elementwise division.
3     y = (1-cos(x))./(x.^2);
4 end
5
6 X = linspace(-1/1e7,1/1e7,1000);
7 Y = f(X);
8
9 plot(X,Y, "LineWidth", 2);
```



approximations.py

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def f(x): return (1-np.cos(x))/x**2
5
6 X = np.linspace(-1e-7, 1e-7, 1000)
7 Y = f(X)
8
9 plt.plot(X, Y)
```



Notice that we've changed the interval from $[-1, 1]$ to $[-10^{-7}, 10^{-7}]$ and found 1000 evenly-spaced points between them; *it was my mistake to suggest such a large interval in the first place!* In any case, we can determine the true value of $f(x)$ as it approaches 0 via

$$\begin{aligned} \lim_{x \rightarrow 0} \frac{1 - \cos(x)}{x^2} &\equiv \lim_{x \rightarrow 0} \frac{\sin(x)}{2x} \\ &= \frac{1}{2} \lim_{x \rightarrow 0} \frac{\sin(x)}{x} \\ &= \frac{1}{2}, \end{aligned}$$

but our plots go wild around $x \approx 1/4 \cdot 10^{-7}$. This is precisely because the values in the numerator and denominator are so close that the computer cannot accurately perform division!

2. Create a function to compute $g(x) = x^3 - x^2$.

- Write down and create functions for the first-order approximation of g at the points $a = 0$, $a = 1/2$, and $a = 1$.
- Do the same for the second-order approximation for g .
- Find the Taylor series for g .

The first- and second-order approximations at a are given by the first and second *Taylor polynomials*:

$$g_1(x) = g(a) + g'(a)(x - a) \quad \text{and} \quad g_2(x) = g(a) + g'(a)(x - a) + g''(a)\frac{(x - a)^2}{2}.$$

Because g is a polynomial, its Taylor series (about a) is just

$$f(x) = g(a) + g'(a)(x - a) + g''(a)\frac{(x - a)^2}{2} + g'''(a)\frac{(x - a)^3}{3!}.$$

Noticing that $g_2(x) = g_1(x) + g''(a)(x - a)^2/2$, we can write the following code to compute their values for arbitrary a :

approximations.m

```

1 function y = g(x)
2     y = x.^3 - x.^2;
3 end
4
5 function y = d1(x)
6     y = 3*x.^2 - 2*x;
7 end
8
9 function y = d2(x)
10    y = 6*x - 2;
11 end
12
13 function y = g1(x, a)
14    y = g(a) + d1(a)*(x-a);
15 end
16
17 function y = g2(x, a)
18    y = g1(x,a) + d2(a)*(1/2)*(x-a).^2;
19 end
20
21 X = linspace(-3, 3, 1000);
22 a = 1/2;
23
24 plot(X, g(X), "LineWidth", 2);
25 hold on
26 plot(X, g1(X, a), "LineWidth", 2);
27 plot(X, g2(X, a), "LineWidth", 2);
28 xlim([-1/2, 3/2]);
29 ylim([-1, 1]);
30 hold off

```

approximations.py

```

1 def g(x):
2     return x**3 - x**2
3
4 def d1(x):
5     return 3*x**2 - 2*x
6
7 def d2(x):
8     return 6*x - 2
9
10 def g1(x,a):
11     return g(a) + d1(a)*(x-a)
12
13 def g2(x,a):
14     return g1(x,a) + (d2(a)*(x-a)**2)/2
15
16 X = np.linspace(-3, 3, 1000)
17 a = 1/2
18
19 # Set viewBox on the plot.
20 plt.ylim(-1,1)
21 plt.xlim(-1/2, 3/2)
22
23 plt.plot(X, g(X))
24 plt.plot(X, g1(X,a))
25 plt.plot(X, g2(X,a))
26 plt.show()

```

In each, we've set $a = 1/2$, which produce the plots below; the original function $g(x)$ is in blue, $g_1(x)$ in orange, and $g_2(x)$ in yellow/green.

