# 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 $x \approx 0$?*

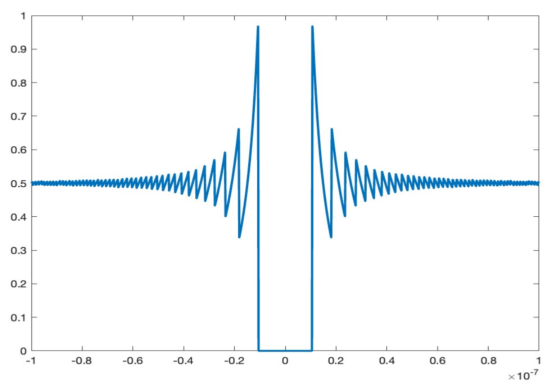
The intent of this exercise was to demonstrate the numerical instability of division $a/b$. Before writing any code, we *expect* that $f(x)$ remains numerically stable as $x$ approaches 0; the below code snippets conduct our experiment.

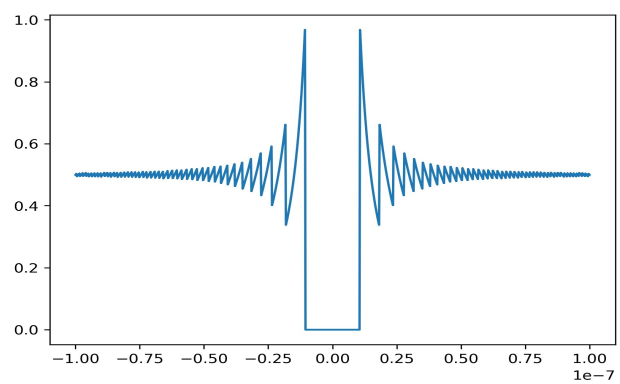
`approximations.m`

```
function y = f(x)
    % Elementwise division.
    y = (1-cos(x))./(x.^2);
end

X = linspace(-1/1e7,1/1e7,1000);
Y = f(X);

plot(X,Y, "LineWidth", 2);
```

`approximations.py`

```
import numpy as np
import matplotlib.pyplot as plt

def f(x): return (1-np.cos(x))/x**2

X = np.linspace(-1e-7, 1e-7, 1000)
Y = f(X)

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

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

but our plots go wild around $x \approx {}^1\!/\!_4 \cdot 10^{-7}$. This is precisley because the values in the numerator and denominator are so close that the computer cannot accurately perform division!

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

   a. *Write down and create functions for the* first-order *approximation of $g$ at the points $a = 0$, $a = 1/2$, and $a = 1$.*
   b. *Do the same for the* second-order *approximation for $g$.*
   c. *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 $a$:

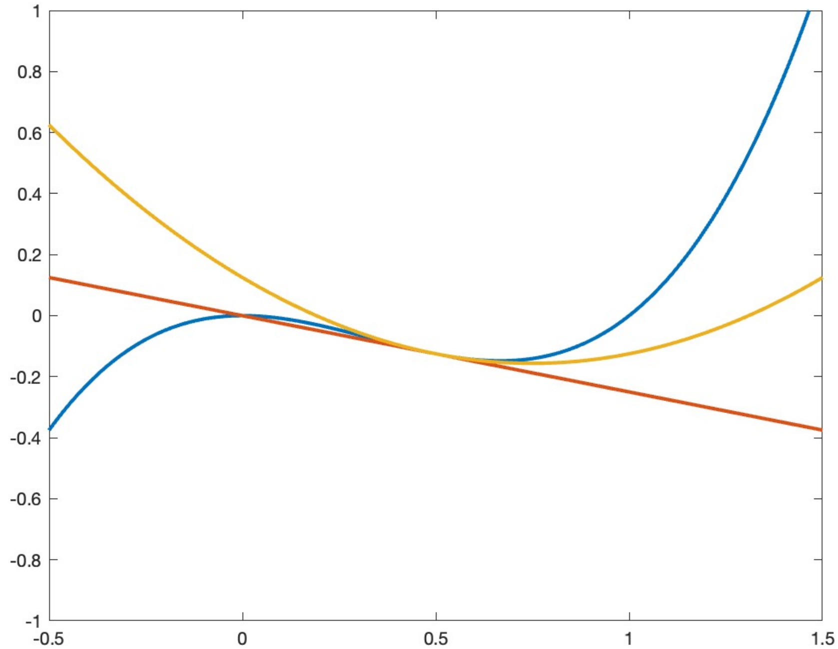
approximations.m

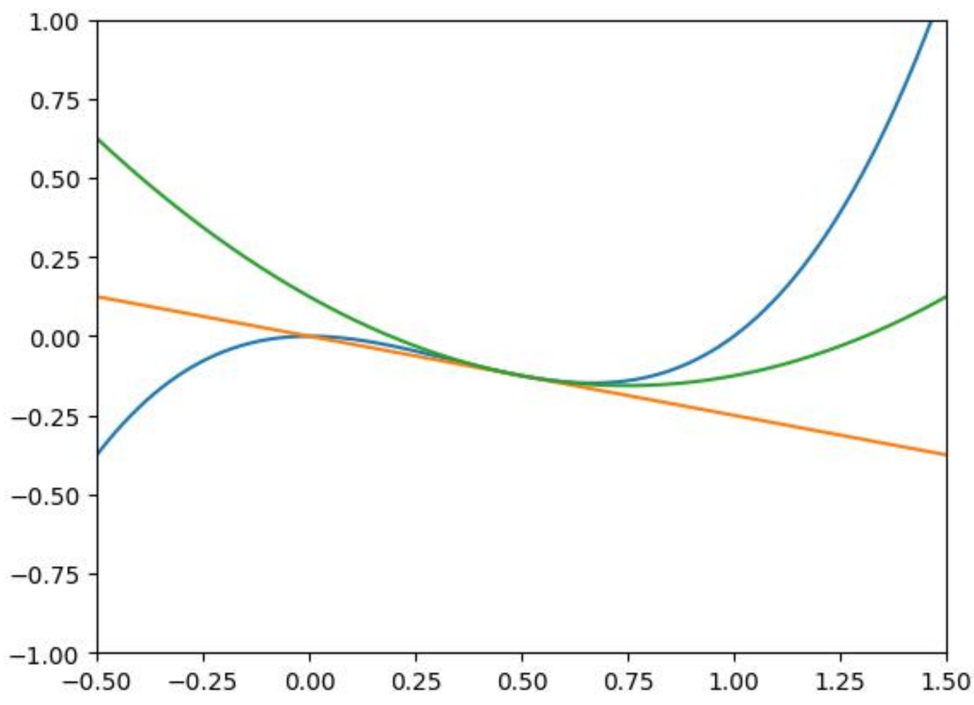
```
function y = g(x)
  y = x.^3 - x.^2;
end

function y = d1(x)
  y = 3*x.^2 - 2*x;
end

function y = d2(x)
  y = 6*x - 2;
end

function y = g1(x, a)
  y = g(a) + d1(a)*(x-a);
end

function y = g2(x, a)
  y = g1(x,a) + d2(a)*(1/2)*(x-a).^2;
end

X = linspace(-3, 3, 1000);
a = 1/2;

plot(X, g(X), "LineWidth", 2);
hold on
plot(X, g1(X, a), "LineWidth", 2);
plot(X, g2(X, a), "LineWidth", 2);
xlim([-1/2,3/2]);
ylim([-1,1]);
hold off
```

approximations.py

```
def g(x):
  return x**3 - x**2

def d1(x):
  return 3*x**2 - 2*x

def d2(x):
  return 6*x - 2

def g1(x,a):
  return g(a) + d1(a)*(x-a)

def g2(x,a):
  return g1(x,a) + (d2(a)*(x-a)**2)/2

X = np.linspace(-3, 3, 1000)
a = 1/2

# Set viewbox on the plot.
plt.ylim(-1,1)
plt.xlim(-1/2, 3/2)

plt.plot(X, g(X))
plt.plot(X, g1(X,a))
plt.plot(X, g2(X,a))
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.


1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-0.5
0
0.5
1
1.5


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
1.50