Please identify anyone, whether or not they are in the class, with whom you discussed your homework. This problem is worth 1 point, but on a multiplicative scale (multiplied by the rest). Collaboration is encouraged.

1. (5 points) Suppose that the sequence $\{x^{(k)}\} \subset \mathbb{R}$ converges to $x^* \in \mathbb{R}$, then we define the "order of convergence" for this sequence if the following limit exists for a $\mu \in \mathbb{R}$ and a $p \geq 1$:

$$\lim_{k\to\infty} \frac{|x^{(k+1)} - x^*|}{|x^{(k)} - x^*|^p} = \mu.$$

   There are three cases:

   1. If $p = 1$ and $\mu = 1$, then this convergence is called "sublinear".
   2. If $p = 1$ and $\mu \in (0, 1)$, then this convergence is called "linear", and μ is called the "rate of convergence".
   3. If $p > 1$ and $\mu \in (0, \infty)$, then this convergence is called "superlinear".

   In the cases of linear convergence and superlinear convergence, p is called the "order of convergence". Note for linear convergence μ has to be strictly less than 1. Determine the sequences as follows, what convergence for each one of them (linear, sublinear, or superlinear), and find the value it converges to, if one has linear convergence, determine its rate of convergence as well.

   (a) $x^{(k)} = 1/k^q$ for a fixed $q \geq 1$.

   (b) $x^{(k)} = \mu^k$ for a fixed $\mu \in (0, 1)$.

   (c) $x^{(k)} = 2^{-2^k}$.

2. (5 points) Consider the function for $\boldsymbol{x} = [x_1, x_2]^\top \in \mathbb{R}^2$

$$f(\boldsymbol{x}) = 3\left(x_1^2 + x_2^2\right) + 4x_1x_2 + 5x_1 + 6x_2 + 7.$$

   Suppose we use a fixed-step-size gradient descent to find the minimizer of f ($\alpha_k = \alpha$ for each k):

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \alpha \nabla f\left(\boldsymbol{x}^{(k)}\right).$$

   Find the largest range of values of α for which algorithm converges for any initial guess $\boldsymbol{x}^{(0)}$ chosen.

3. (5 points) Consider the function $f : \mathbb{R} \to \mathbb{R}$ given by $f(x) = \frac{1}{2}(x - c)^2$ for $c \in \mathbb{R}$. Use the gradient descent algorithm to compute the minimizer of f we have: for a step size $\alpha_k \in (0, 1)$ at each iteration $k = 0, 1, 2, \ldots$

$$x^{(k+1)} = x^{(k)} - \alpha_k f'\left(x^{(k)}\right).$$

   (a) Derive a formula of $f\left(x^{(k+1)}\right)$ in terms of $f\left(x^{(k)}\right)$ and α_k. (HINT: simply plug in the iterative formula to f)

(b) Show that the gradient descent converges regardless of the initial guess $x^{(0)}$ if $\sum_{k=0}^{\infty} \alpha_k = \infty$. (HINT: you can directly use the fact that for any sequence $\{\alpha_k\} \subset (0,1)$, we have $\prod_{k=0}^{\infty}(1-\alpha_k) = 0 \Leftrightarrow \sum_{k=0}^{\infty} \alpha_k = \infty$. If you want to prove this, you can prove by contradiction of taking the logarithm on both sides.)

4. (5 points) Consider the vanila linear regression problem with m samples and n features, that is for $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, $m \geq n$, and $\boldsymbol{b} \in \mathbb{R}^m$, one oughts to solve the following minimization problem

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}) := \min_{\boldsymbol{x}\in\mathbb{R}^m} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|^2 = \min_{\boldsymbol{x}\in\mathbb{R}^m} (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})^\top (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})$$

   (a) Write down the ∇f and $\nabla^2 f$. (Homework 1's result can be used).

   (b) Write down the explicit fixed-step-size gradient algorithm, involving only $\boldsymbol{x}^{(k+1)}$, $\boldsymbol{x}^{(k)}$, $\boldsymbol{A}$, and $\boldsymbol{b}$, for solving this optimization problem using the result in part (a).

   (c) Suppose that $\boldsymbol{A} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$ with $\lambda_2 > \lambda_1 \geq 1$. Find the largest range of values for α such that the algorithm in part (b) converges to the minimizer.

5. (5 points) Read the manual and the example featured in `torch.nn.bilinear`: https://pytorch.org/docs/stable/generated/torch.nn.Bilinear.html (a screencap in Fig. 1 is attached at the end of this pdf file). For the inputs of `nn.bilinear`, the first dimension of inputs variables in $*$ is usually the "batch" (except in Attention) and is often refered to as the "batch dimension", indicating the number of samples stored in this tensor. The second dimension in the inputs is often referred to as the "feature dimension". The extra "out_features =: c_o" dimension can be viewed as a stack of c_o matrices, for example, if the weight tensor A has "out_features = 2", then $A = [A^{(1)}, A^{(2)}]$, and the bilinear inner product is computed as $[x_1^\top A^{(1)} x_2 + b, x_1^\top A^{(2)} x_2 + b]$. Answer the following questions:

   (a) What are the "feature" dimensions of x_1, x_2 in the example given in the manual?

   (b) Compute by hand, that if the transformation `m = nn.Bilinear(3, 3, 2)` with the weight matrix A set to $A = [3I, -2I]$ where I is the identity matrix, the bias vector b is zero, and let $x = [1/2, -1, 1/2]^\top$. What is the output of `m(x, x)`? Equivalently, you can test your answer by using directly from `torch.nn.functional.bilinear` for which the `nn.bilinear` is a wrapper (https://pytorch.org/docs/stable/generated/torch.nn.functional.bilinear.html).

```
x = torch.Tensor([1/2, -1, 1/2])
m = torch.stack([3*torch.eye(3), -2*torch.eye(3)], dim=0)
torch.nn.functional.bilinear(x, x, m)
```

BILINEAR

CLASS torch.nn.Bilinear(*in1_features*, *in2_features*, *out_features*, *bias=True*, *device=None*, *dtype=None*) [SOURCE]

Applies a bilinear transformation to the incoming data: $y = x_1^T A x_2 + b$

Parameters:

- **in1_features** (*int*) – size of each first input sample
- **in2_features** (*int*) – size of each second input sample
- **out_features** (*int*) – size of each output sample
- **bias** (*bool*) – If set to False, the layer will not learn an additive bias. Default: `True`

Shape:

- Input1: $(*, H_{in1})$ where $H_{in1} = \text{in1\_features}$ and $*$ means any number of additional dimensions including none. All but the last dimension of the inputs should be the same.
- Input2: $(*, H_{in2})$ where $H_{in2} = \text{in2\_features}$.
- Output: $(*, H_{out})$ where $H_{out} = \text{out\_features}$ and all but the last dimension are the same shape as the input.

Variables:

- **weight** (*torch.Tensor*) – the learnable weights of the module of shape $(\text{out\_features}, \text{in1\_features}, \text{in2\_features})$. The values are initialized from $\mathcal{U}(-\sqrt{k}, \sqrt{k})$, where $k = \frac{1}{\text{in1\_features}}$
- **bias** – the learnable bias of the module of shape (out_features). If `bias` is `True`, the values are initialized from $\mathcal{U}(-\sqrt{k}, \sqrt{k})$, where $k = \frac{1}{\text{in1\_features}}$

Examples:

```
>>> m = nn.Bilinear(20, 30, 40)
>>> input1 = torch.randn(128, 20)
>>> input2 = torch.randn(128, 30)
>>> output = m(input1, input2)
>>> print(output.size())
torch.Size([128, 40])
```

Figure 1: Documentation drawn from PyTorch.