H05: Introduction to Optimization

Faculty of Engineering, Khon Kaen University

Submission: https://autolab.en.kku.ac.th

===

* Submit an answer to a question with a file with `txt` extension. E.g., an answer for Q1 should be submitted in a text file "`Q1.txt`"

* Submit a program to a (programming) problem with a file with a proper extension. E.g., a python program for P2 should be named "`P2.py`"

* Each question or problem is worth 840 points.

* Floating-point numbers are graded using tolerance 0.001.

===

Q1. Optimization. Given *g(u)* is only defined as shown in Table 1, answer the following sub-questions. Assume *g(u)* is undefined beyond the table below.

Table 1. Function g(u) values.

u	*g(u)*	
-5	13	
-4.5	14	
-4	13	
-3.5	9	
-3	4	
-2.5	-4	
-2	0	
-1.5	-2	
-1	-8	
-0.5	-14	
0	-19	
0.5	-21	
1	-8	
1.5	-2	
2	0.5	
2.5	4	
3	5	
3.5	9	
4	11	
4.5	10	

5	7	

Q1.1. Find $v = argmin_u\ g(u)$, the optimal value of the objective function (OVO), and $g(v)$.

Q1.2. Find $v = argmin_u\ |g(u)|$, OVO, and $g(v)$.

Q1.3. Find $v = argmax_u\ g(u)$, OVO, and $g(v)$.

Q1.4. Find $v = argmax_u\ g(-u)$, OVO, and $g(v)$.

Q1.5. Find $v = argmax_u - g(u)$, OVO, and $g(v)$.

Q1.6. Find $v = argmax_u\ 5 - g(u)$, OVO, and $g(v)$.

Q1.7. Find $v = argmax_u\ g(u-1)$, OVO, and $g(v)$.

Q1.8. Find $v = argmax_u\ g(0.5-u)$, OVO, and $g(v)$.

Q1.9. Find $v = argmax_u - g(0.5-u)$, OVO, and $g(v)$.

Q1.10. Find $v = argmin_u|\ 0.5 - g(u)\ |$, OVO, and $g(v)$.

Q1.11. Find $v = argmin_u\ g(u)$ s.t. $u < -1$, OVO, and $g(v)$.

Q1.12. Find $v = argmax_u\ g(u)$ s.t. $-2.5 \le u < 2$, OVO, and $g(v)$.

Write your answers in the following format.

```
Q1.1. v = 0 ; OVO = 0 ; g(v) = 0
Q1.2. v = 0 ; OVO = 0 ; g(v) = 0
Q1.3. v = 0 ; OVO = 0 ; g(v) = 0
Q1.4. v = 0 ; OVO = 0 ; g(v) = 0
Q1.5. v = 0 ; OVO = 0 ; g(v) = 0
Q1.6. v = 0 ; OVO = 0 ; g(v) = 0
Q1.7. v = 0 ; OVO = 0 ; g(v) = 0
Q1.8. v = 0 ; OVO = 0 ; g(v) = 0
Q1.9. v = 0 ; OVO = 0 ; g(v) = 0
Q1.10. v = 0 ; OVO = 0 ; g(v) = 0
Q1.11. v = 0 ; OVO = 0 ; g(v) = 0
Q1.12. v = 0 ; OVO = 0 ; g(v) = 0
```

Q2. Multi-dimensional optimization. Given $g(\boldsymbol{u})$ is only defined as shown in Table 2, answer the following sub-questions.

Table 2. Function g(u) values, e.g., g(u=(-4,-3)) = 12, g(u=(0,0)) = 2.1, and so on.

$\boldsymbol{u} = (u_1, u_2)$	u_2				
u_1	**-5**	**-3**	**0**	**3**	**5**

-8	5	6	7.2	0.4	-0.11
-4	7	12	8	-0.3	-2.3
0	5.4	4	2.1	-17	-9
4	3.2	0	-9	-24	-16
8	2.1	1.9	0.4	-1	-2

Q2.1. Find $\boldsymbol{v} = argmin_{\boldsymbol{u}}\, g(\boldsymbol{u})$, the optimal value of the objective function (OVO), and $g(\boldsymbol{v})$.

Q2.2. Find $\boldsymbol{v} = argmin_{\boldsymbol{u}}\, |g(\boldsymbol{u})|$, OVO, and $g(\boldsymbol{v})$.

Q2.3. Find $\boldsymbol{v} = argmax_{\boldsymbol{u}}\, g(\boldsymbol{u})$, OVO, and $g(\boldsymbol{v})$.

Q2.4. Find $\boldsymbol{v} = argmax_{\boldsymbol{u}}\, g(-\boldsymbol{u})$, OVO, and $g(\boldsymbol{v})$.

Q2.5. Find $\boldsymbol{v} = argmin_{\boldsymbol{u}}\, (g(\boldsymbol{u}) - 9)^2$, OVO, and $g(\boldsymbol{v})$.

Write your answers in the following format.

```
Q2.1. v = ( 0 , 0 ) ; OVO = 0 ; g(v) = 0
Q2.2. v = ( 0 , 0 ) ; OVO = 0 ; g(v) = 0
Q2.3. v = ( 0 , 0 ) ; OVO = 0 ; g(v) = 0
Q2.4. v = ( 0 , 0 ) ; OVO = 0 ; g(v) = 0
Q2.5. v = ( 0 , 0 ) ; OVO = 0 ; g(v) = 0
```

Q3. Operating point and maximal power. Suppose the I-V curve of a solar panel is as shown in Figure 1, to maximize the benefit, we want to find the best operating point: $v^* = argmax_v P(v)$

where v represents operating voltage and P is an output power. The best operating point is the operating voltage where the corresponding power is at its maximum.

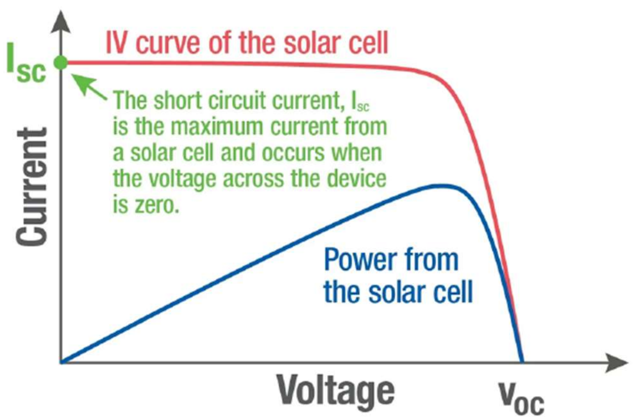


Figure 1. I-V curve of a solar panel.

Suppose power as a function of the operating voltage can be estimated from

$$P(v) = 5\,v - 3\,(1 - \ln(3.8 - v)) - 1$$

Equation 1

as shown in Figure 2, where v represents the operating voltage (within the range between 0 and 3.7), answer the following questions.

Q3.1. What is the best operating point $v^* = argmax_{v\in(0,3.7)}\,P(v)$?And, what maximal power can it deliver?

Q3.2. If this solar panel is to supply a load requirement of 10 watt, would it be sufficient for the task? If not how many panels do we need at least to supply such a load? Suppose every solar panel is identical.

Q3.3. If this solar panel is to supply a load requirement of 20 watt, would it be sufficient for the task? If not how many panels do we need at least to supply such a load? Suppose every solar panel is identical.

Q3.4. If this solar panel is to supply a load requirement of 20 watt with safety margin of 20%, would it be sufficient for the task? If not how many panels do we need at least to supply such a load? Suppose every solar panel is identical.

Hint. (1) You are free to use any mean to find the answer for Q3.1, but P6 may be handy.

(2) In Q3.4, to have 20% margin, it means that 20-watt load must be supplied by a power supplier with capability of at least 24-watt.

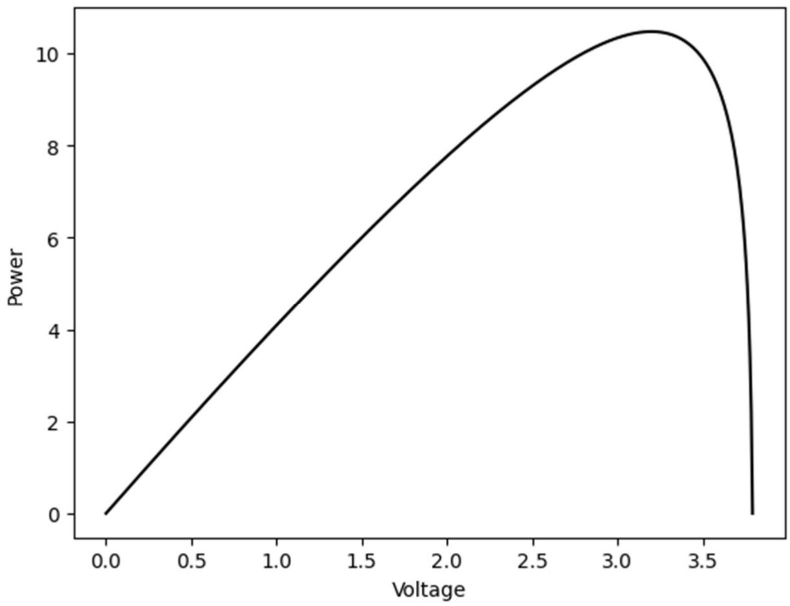


Figure 2

Write your answers in the following format.

```
Q3.1. v = 0 volt and P(v) = 0 watt
Q3.2. Yes/No. We need 0 panel(s).
Q3.3. Yes/No. We need 0 panel(s).
Q3.4. Yes/No. We need 0 panel(s).
```

Grader guide: autograder uses 0.001 tolerance. Double-check your answer:

$P(v^*) > P(v \neq v^*)$. This means $P(v^*) > P(v^* - 0.001)$ and $P(v^*) > P(v^* + 0.001)$. Of course, you can test more cases. Better test, better answer!

P4. Objective function. Given Q3, write function `power(v)` whose argument `v` takes operating voltage and the function returns the power output, per Equation 1.

Hint. Numpy `log` does natural logarithm.

Given the functions are called as follows

```
from P4 import power

if __name__ == "__main__":
    v = 2.8
    p = power(v)
    print('power=', p)
```

output example may look like:

```
power= 10.0
```

P5. Minimization with gradient descend algorithm. Given the problem formulation: $min_u\ (u - 1.5)^2 - 3\ln(u + 2)$.

Write functions `loss`, `grad`, and `minimizer`.

(1) Function `loss` takes argument `u` and return $(u - 1.5)^2 - 3\ln(u + 2)$.

(2) Function `grad` takes argument `u` and return the gradient of the loss function. Recall that given loss, $L = (u - 1.5)^2 - 3\ln(u + 2)$, gradient $\nabla_u L = \frac{dL}{du}$; $\frac{d\ln(x)}{dx} = \frac{1}{x}$; and $\frac{d\ln(u)}{dx} = \frac{1}{u}\frac{du}{dx}$.

(3) Function `minimizer` takes an initial value for `u`, step size, and a number of steps, then runs gradient descend algorithm for the specified hyperparameters and returns the minimizer as a result.

Given the functions are called as follows

```
from P5 import loss, grad, minimizer

if __name__ == "__main__":
    u = 1
    print('loss=', loss(u))
    print('grad=', grad(u))
    print('minimizer=', minimizer(u, 0.1, 10))
    print('minimizer=', minimizer(u, 0.3, 10))
    print('minimizer=', minimizer(u, 0.9, 10))
    print('minimizer=', minimizer(u, 0.3, 100))
```

output example may look like:

```
loss= -3.045836866004329
grad= -2.0
minimizer= 1.8140595939724171
minimizer= 1.8859837452059862
minimizer= 1.113595035619058
minimizer= 1.8860009363293828
```

P6. Maximization with gradient descend algorithm. Given the problem formulation:

$$max_v \, P(v)$$

where $P(v) = 5\, v - 3\,(1 - \ln(3.8 - v)) - 1.$

Write functions `objective`, `grad`, and `maximizer`.

(1) Function `objective` takes argument `v` and return $P(v)$.

(2) Function `dPv` takes argument `v` and return the gradient of the objective function, $\frac{dP}{dv}$.

(3) Function `maximizer` takes an initial value for `v`, step size, and a number of steps, then runs gradient descend algorithm for the specified hyperparameters and returns the maximizer as a value of the maximizer found in the process.

Hint. (1) It is MAXIMIZATION! students are encouraged to use visualization to gain understanding as much as possible. (But do not put visualization in your submission. Grading environment does not have visualization tools.)

(2) Recall Q3.
(3) Recall basic calculus as in P6.

Given the functions are called as follows

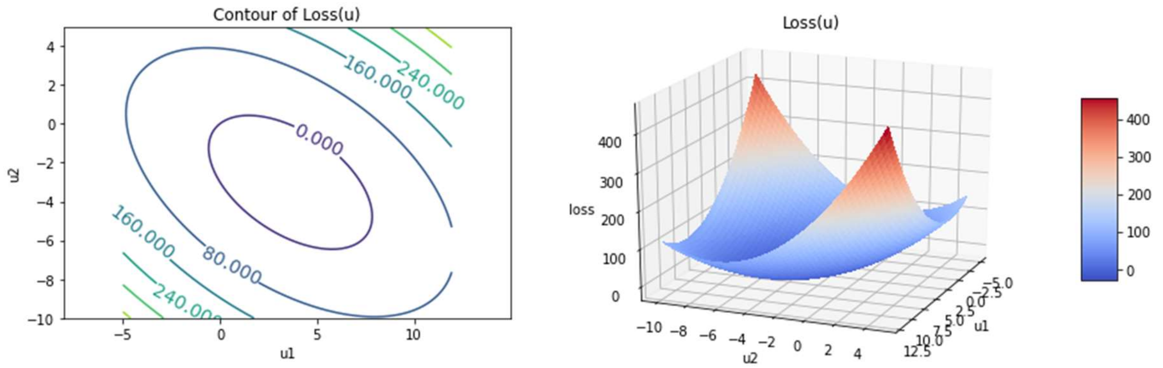
```
from P6 import objective, dPv, maximizer

if __name__ == "__main__":
    v = 1.2
    print('objective=', objective(v))
    print('dPv=', dPv(v))
    print('v*=', maximizer(v, 0.001, 100))
    print('v*=', maximizer(v, 0.01, 10))
    print('v*=', maximizer(v, 0.02, 10))
```

output example may look like:

```
objective= 4.866534335082308
dPv= 3.846153846153846
v*= 1.5753988986418308
v*= 1.5762509470308657
v*= 1.932528959003603
```

P7. Two-dimensional problem. Given a loss $L(\vec{u}) = \vec{u}^T \cdot \boldsymbol{A} \cdot \vec{u} + \vec{b}^T \cdot \vec{u}$, where $\boldsymbol{A} = \begin{bmatrix} 2 & 1 \\ 1.5 & 3 \end{bmatrix}$ and $\vec{b} = \begin{bmatrix} -7 \\ 9 \end{bmatrix}$. Use gradient descend algorithm to solve $\min_{\vec{u}} L(\vec{u})$. Figure 3 visualizes the loss function.



a b

Figure 3. $L(\vec{u})$ in contour plot (a) and perspective plot (b).

Write functions `loss`, `grad`, and `minimizer`.

(1) Function `loss` takes argument `uvec` as a numpy array of shape (2,1) and return value of $L(\vec{u})$ as a python float.

(2) Function `grad` takes argument `uvec` as a numpy array of shape (2,1) and return the gradient $\nabla_{\vec{u}} L(\vec{u})$ as a numpy array of shape (2,1).

(3) Function `minimizer` takes initial value for `uvec` as a numpy array of shape (2,1), step size, and a number of steps, then runs gradient descend algorithm for the specified hyperparameters and returns the result as a numpy array of shape (2,1).

<u>Hint</u>

(1) Work out the math. For example,

$$g(\vec{x}) = \vec{x}^T \cdot \vec{x} + [5 \quad 6] \cdot \vec{x} = x_1^2 + x_2^2 + 5\,x_1 + 6\,x_2.$$

Hence, $$\nabla_{\vec{x}} g = \begin{bmatrix} \frac{\partial g}{\partial x_1} \\ \frac{\partial g}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 2\,x_1 + 5 \\ 2\,x_2 + 6 \end{bmatrix}.$$

(2) Method `np.array.reshape` may be handy. See examples below.

Code	Outcome on screen

`v1 = 4`
`v2 = 5`
`dL = np.r_[v1, v2].reshape((2,-1))`
`print(dL)`
	`[[4]`
	` [5]]`

`v1 = [4, 8, 16]`
`v2 = [5, 10, 20]`
`dL = np.r_[v1, v2].reshape((2,-1))`
`print(dL)`
	`[[ 4  8 16]`
	` [ 5 10 20]]`

Method `reshape` is to force `np.array` to have the specific shape, i.e., `2` in the first dimension and leave the second dimension adjusted. Allowing a flexible dimension is convenient and a proper use can make code more robust.

Invocation example

Run:

Test P7

```
import numpy as np
from P7 import loss, grad, minimizer

if __name__ == '__main__':
    u0 = np.array([[0], [0]])
    uz = minimizer(u0, lr=0.1, N=500)
    print('type=', type(uz))
    print('shape=', uz.shape)
    print(':  u*=', uz)
    print(':  loss(u*)=', loss(uz))
    print(':  grad(u*)=', grad(uz))
```

Screen:

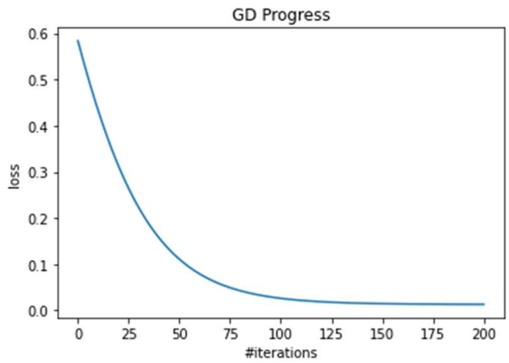
```
type= <class 'numpy.ndarray'>
shape= (2, 1)
```

```
:  u*= [[ 3.63380282]
 [-3.01408451]]
:  loss(u*)= -26.281690140845075
:  grad(u*)= [[-8.88178420e-16]
 [ 1.77635684e-15]]
```

Again, notice that for unconstrained optimization gradient is “zero” (or close to zero or a zero vector) at the optimum.

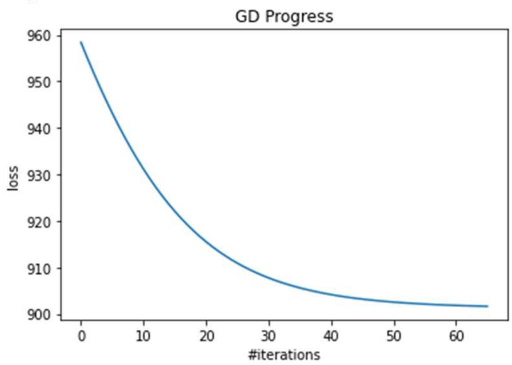
Q8. Gradient-descend progress diagnosis. Answer the following questions.

Q8.1. Given the following progress from running a gradient descend algorithm, what should we do next?



(a) It could have reached the minimizer. We can stop and check the obtained solution.
(b) It seems like that the step size is too large. Make a step size smaller and run it longer.
(c) It seems like that the step size is much too large. Re-initialize the variables, make a step size much smaller and run it again.

Q8.2. Given the following progress from running a gradient descend algorithm with a single value of step size, what should we do next?

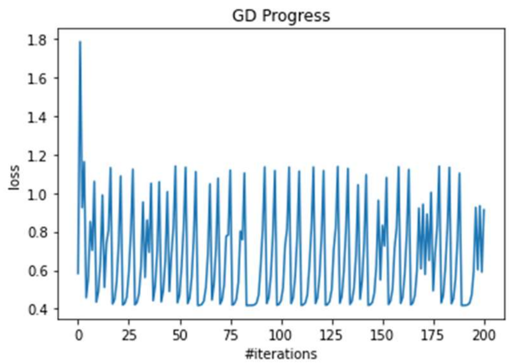


(a) It seems like that the step size is much too large. Re-initialize the variables, make a step size much smaller and run it again.
(b) It is better to keep it run a litter longer and see how it will go.
(c) It seems like something wrong. Stop and double-check the gradient.

Q8.3. Given the following progress from running a gradient descend algorithm, what should we do next?

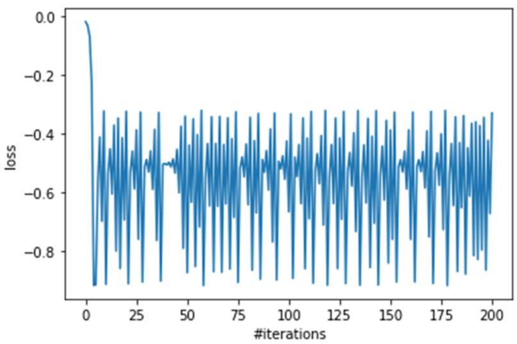
(a) It seems like that the step size is much too large. Re-initialize the variables, make a step size much smaller and run it again.
(b) It seems like something wrong. Stop and double-check the gradient.
(c) It seems like a number of steps is just too small. Let it run longer.

Q8.4. Given the following progress from running a gradient descend algorithm, what should we do next?



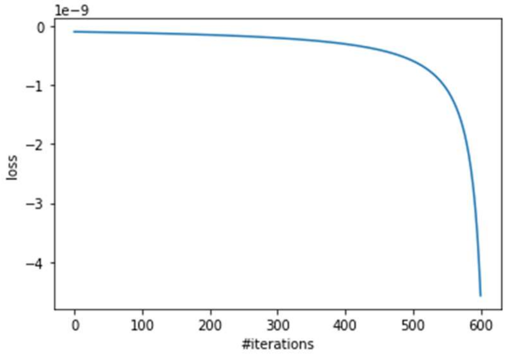
(a) It seems like a number of steps is just too small. Let it run longer.
(b) It seems like a step size is just too small. With a larger step size, run it longer or re-start.
(c) It seems like a step size is just too large. With a smaller step size, run it longer or re-start.

Q8.5. Given the following progress from running a gradient descend algorithm, what should we do next?



(a) It seems like a step size is just too small. With a larger step size, run it longer or re-start.
(b) It seems like a step size is just too large. With a smaller step size, run it longer or re-start.
(c) It seems like a number of steps is just too small. Let it run longer.

Q8.6. Given the following progress from running a gradient descend algorithm, what should we do next?



(a) Continue running with a smaller step size.
(b) Continue running with a larger step size.
(c) Restart it with a much smaller step size.

Write your answers in the following format.

```
Q8.1. Answer = ?a/b/c?
Q8.2. Answer = ?a/b/c?
Q8.3. Answer = ?a/b/c?
Q8.4. Answer = ?a/b/c?
Q8.5. Answer = ?a/b/c?
Q8.6. Answer = ?a/b/c?
```