
گروه ۴
مهرداد قادری
یگانه محسنی
کیمیا یوسفی
فاطمه زهرا خاسب
کاظم ملکی


بسم الله الرحمن الرحیم

جدول پارامتر ها

Table 1. Parameter values for microengine reliability analysis

Parameters	*Values*	*Sources*
H	0.00125 μm^3	Tanner and Dugger (2003)
D	1.5 GPa (for polysilicon material used for springs)	Tanner and Dugger (2003)
φ	0	Tanner and Dugger (2003)
β	$\sim N(\mu_\beta, \sigma_\beta^2)$ $\mu_\beta = 8.4823 \times 10^{-9}$ μm^3 and $\sigma_\beta = 6.0016 \times 10^{-10}$ μm^3	Tanner and Dugger (2003) Peng *et al.* (2009a)
λ	2.5×10^{-5}	Assumption
Y_i	$\sim N(\mu_Y, \sigma_Y^2)$ for $i = 1, 2, \ldots, \infty$ $\mu_Y = 1 \times 10^{-4}$ μm^3 and $\sigma_Y = 2 \times 10^{-5}$ μm^3	Assumption
W_i	$\sim N(\mu_W, \sigma_W^2)$ for $i = 1, 2, \ldots, \infty$ $\mu_W = 1.2$ GPa and $\sigma_W = 0.2$ GPa	Assumption

```
z= 125*1e-11 # Degradation thereshold
def state(amount):


  if (amount<z):
    state=0       #work

  if (amount >z):
    state=1 # fail

  return state
```

MDP(Marcov decision process):

- State(در MDP, State ها به صورت گسسته هستند و مدل فرسایش ما پیوسته است پس با سطح بندی آن را گسسته سازی میکنیم)

علت کاهش تقسیم بندی وضعیت ها ؟

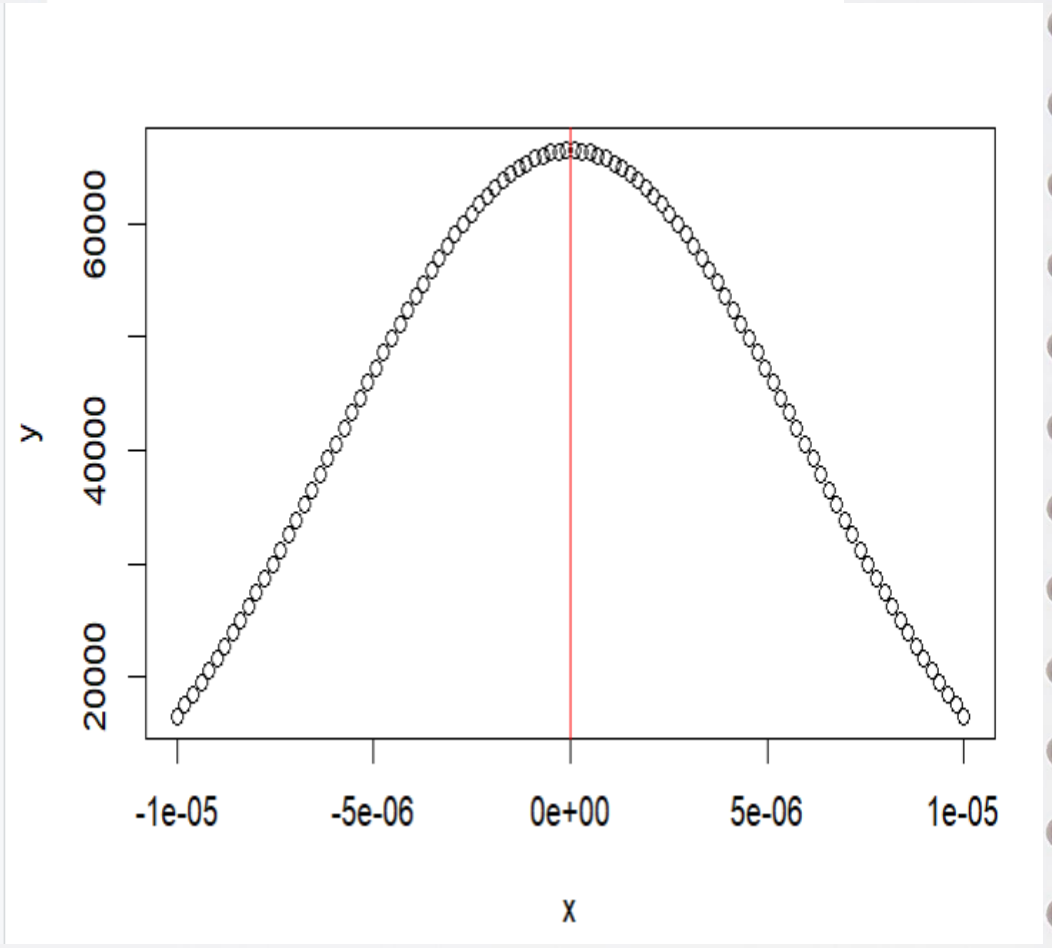
```
mu_y=1e-10
sigma_y=2*(1e-11)
mu_beta=8.4823*(1e-15)
sigma_beta=6.0016*1e-16
mu_sum=mu_y+(t*mu_beta)
sigma_sum=sigma_y+((t^2)*sigma_beta)
t=1*(10^5)
print(mu_sum)
print(sigma_sum)
a=mu_sum-(3*sigma_sum)
b=mu_sum+(3*sigma_sum)
print(a)
print(b)
x=seq(-1e-5,1e-5,length=100)
length(x)
y=dnorm(x,mu_sum,sigma_sum)
plot(x,y)

abline(v=125*1e-11,col = "gray60")
abline(v=9.4823e-10,col = "red")
```

MDP(Marcov decision process):

- State(در MDP, State ها به صورت گسسته هستند و مدل فرسایش ما پیوسته است پس با سطح بندی آن را گسسته سازی میکنیم)

علت کاهش تقسیم بندی وضعیت ها ؟



MDP(Marcov decision process):

- State(در MDP, State ها به صورت گسسته هستند و مدل فرسایش ما پیوسته است پس با سطح بندی آن را گسسته سازی میکنیم)

علت کاهش تقسیم بندی وضعیت ها ؟

```
cost=np.matrix([[-1,-11],[-51,-61]])
cost
```

```
matrix([[ -1, -11],
        [-51, -61]])
```

MDP(Marcov decision process):

- ماتریس پاداش ها (هزینه ها)

نکته :

- هزینه بازرسی ۱ دلار
- هزینه تعویض ۱۰ دلار
- هزینه جریمه ۵۰ دلار

```
z=125*1e-11
mu_beta=8.4823*1e-15
sigma_beta=6.0016*1e-16
mu_y=1e-10
sigma_y=2*1e-11
lamda_shock=2.5*1e-5
def Environment(Degradation,action,tov):
    if (Degradation >= z) and (action==0):
        next_state=1
        reward=cost[next_state,action]
        Degradation =Degradation

    if (Degradation < z and action==0): #do nothing

        n=np.random.poisson(lamda_shock*tov,1)#number of efective shock
        #print(n)
        w=np.random.normal(1.2,0.2,int(n))
        if any([x>1.5 for x in w]) :
                next_state=1
                reward=cost[next_state,action]
                Degradation =z
        else:
                y=np.random.normal(mu_y,sigma_y,int(n))
                magnitude = sum(y)
                Degradation = Degradation+ magnitude + (np.random.normal(mu_beta,sigma_beta) * tov)
                next_state=state(Degradation)
                reward=cost[next_state,action]

    if ( action==1 ): #replace

        n=np.random.poisson(lamda_shock*tov,1)#number of efective shock
        #print(n)
        w=np.random.normal(1.2,0.2,int(n))
        if any([x>1.5 for x in w]) :
                next_state=1
                reward=cost[next_state,action]
                Degradation =z
        else:
                y=np.random.normal(mu_y,sigma_y,int(n))
                magnitude = sum(y)
                Degradation = magnitude + (np.random.normal(mu_beta,sigma_beta) * tov)
                next_state=state(Degradation)
                reward=cost[next_state,action]



    return (next_state,reward,Degradation)
```

MDP(Marcov decision process):

- محیط

```
z=125*1e-11
mu_beta=8.4823*1e-15
sigma_beta=6.0016*1e-16
mu_y=1e-10
sigma_y=2*1e-11
lamda_shock=2.5*1e-5
def Environment(Degradation,action,tov):
    if (Degradation >= z) and (action==0):
        next_state=1
        reward=cost[next_state,action]
        Degradation =Degradation

    if (Degradation < z and action==0): #do nothing

        n=np.random.poisson(lamda_shock*tov,1)#number of efective shock
        #print(n)
        w=np.random.normal(1.2,0.2,int(n))
        if any([x>1.5 for x in w]) :
                next_state=1
                reward=cost[next_state,action]
                Degradation =z
        else:
                y=np.random.normal(mu_y,sigma_y,int(n))
                magnitude = sum(y)
                Degradation = Degradation+ magnitude + (np.random.normal(mu_beta,sigma_beta) * tov)
                next_state=state(Degradation)
                reward=cost[next_state,action]

    if ( action==1 ): #replace

        n=np.random.poisson(lamda_shock*tov,1)#number of efective shock
        #print(n)
        w=np.random.normal(1.2,0.2,int(n))
        if any([x>1.5 for x in w]) :
                next_state=1
                reward=cost[next_state,action]
                Degradation =z
        else:
                y=np.random.normal(mu_y,sigma_y,int(n))
                magnitude = sum(y)
                Degradation = magnitude + (np.random.normal(mu_beta,sigma_beta) * tov)
                next_state=state(Degradation)
                reward=cost[next_state,action]



    return (next_state,reward,Degradation)
```

```python
def choose_action(epsilon, state):
    if (np.random.random() < epsilon):
        return random.choice([0,1])
    else:
        return np.argmax(Q_table[state])
```

MDP(Marcov decision process):

- انتخاب اقدامات

نکته : بر اساس الگوریتم ∈گریدی

Q-Learning Algorithm for Off-Policy Control

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$

Repeat (for each episode):

  Initialize S

  Repeat (for each step of episode):

    Choose A from S using policy derived from Q (e.g., ε-greedy)

    Take action A, observe R, S'

    $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$

    $S \leftarrow S'$;

  until S is terminal

MDP(Marcov decision process):

- الگوریتم Q-Learning

مراحل الگوریتم

ε-Greedy Exploration

- Simplest idea for ensuring continual exploration
- All m actions are tried with non-zero probability
- With probability $1-\epsilon$ choose the greedy action
- With probability ϵ choose an action at random

$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a^* = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q(s,a) \\ \epsilon/m & \text{otherwise} \end{cases}$$

MDP(Marcov decision process):

- انتخاب اقدامات

نکته : بر اساس الگوریتم ∈گریدی

```python
#my_score=[]
tau_list = [1.05*(10**5),1.1*(10**5),1.15*(10**5),1.2*(10**5),1.25*(10**5),1.3*(10**5),1.35*(10**5),1.4*(10**5),1.45*(10**5),1.5*(10**5)]
q_table_list=[]
for t in tau_list:
    tov=t
    epsilon = 1
    learning_rate = 1
    discount = 0.5
    Degradation=0
    Q_table = np.zeros((2,2)) #1

    scores = []


# Looping for each episode
    for e in range(1000): #2
    # Initializes the state
        current_state = 0 #it is new #3

        rewards = []


    # Looping for each step
        for j in range(int(40000000//tov)):#4
        # Choose A from S
            action = choose_action(epsilon,current_state) #5
        #print(action)
        # Take action
            next_state ,immediate_reward,Degradation = Environment(Degradation,action,tov) #6
            rewards.append(immediate_reward)
            new_state = next_state

        # Update Q(S,A)
            Q_table[current_state][action] += (learning_rate *
                                       (immediate_reward
                                        + discount * np.max(Q_table[new_state])
                                        - Q_table[current_state][action])) #7


            current_state = new_state #8

        learning_rate *= 0.99
        epsilon *= 0.99
        scores.append(sum(rewards))
    #my_score.append(mean(scores))
    print(t)
    #print(Q_table)
    q_table_list.append(Q_table)
```

MDP(Marcov decision process):

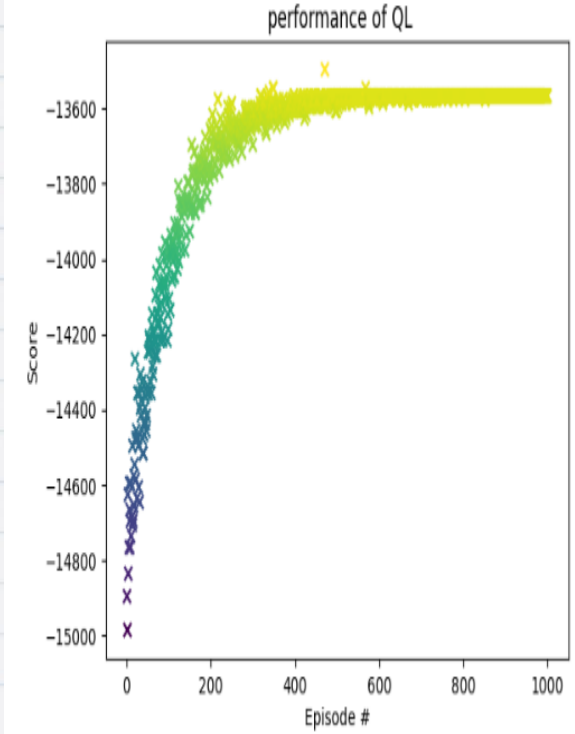
- به دست آوردن ماتریس Q-Table

نکته : ماتریس بهینه

```
] tau_list = [1.05*(10**5),1.10*(10**5),1.15*(10**5),1.20*(10**5),1.25*(10**5),1.30*(10**5),1.35*(10**5),1.40*(10**5),1.45*(10**5),1.50*(10**5)]

for t in tau_list:
    fig = plt.figure()
    ax = fig.add_subplot(111)
    x = np.arange(1, len(scores) + 1)
    y = scores
    plt.scatter(x, y, marker='x', c=y)
    plt.ylabel('Score')
    plt.xlabel('Episode #')
    plt.title(' performance of QL')
    print("this plot is for",t)
    plt.show()
```


```
this plot is for 145000.0
```



MDP(Marcov decision process):

- به دست آوردن ماتریس Q-Table

نکته : همگرایی الگوریتم

```
tau_list = [1.05*(10**5),1.10*(10**5),1.15*(10**5),1.20*(10**5),1.25*(10**5),1.30*(10**5),1.35*(10**5),1.40*(10**5),1.45*(10**5),1.50*(10**5)]


score_tow=[]
for i in range(10):
    tov=tau_list[i]
    #epsilon = 0
    #learning_rate = 0
    discount = 0.5
    Degradation=0
    #Q_table = np.zeros((5,2)) #1

    scores = []


# Looping for each episode
    for e in range(1000): #2
    # Initializes the state
        current_state = 0 #it is new #3

        rewards = []


    # Looping for each step
        for j in range(int(40000000//tov)):#4
        # Choose A from S
            nqtable = q_table_list[i]
            action = np.argmax(nqtable[current_state])
            #for i in range(10):
             #   nqtable=q_table_list[i]
              #  action = np.argmax(nqtable[current_state])
               # print(action)
            #5
        #print(action)
        # Take action
            next_state ,immediate_reward,Degradation = Environment(Degradation,action,tov) #6
            rewards.append(immediate_reward)
            new_state = next_state

        # Update Q(S,A)
            #Q_table[current_state][action] += (learning_rate *
                                        #(immediate_reward
                                         #+ discount * np.max(Q_table[new_state])
                                         #- Q_table[current_state][action])) #7

            current_state = new_state #8

        #learning_rate *= 0.99
        #epsilon *= 0.99
        scores.append(sum(rewards))
    score_tow.append(mean(scores))
    #my_score.append(mean(scores))
    print(tov)
    #print(Q_table)
    #q_table_list.append(Q_table)
```

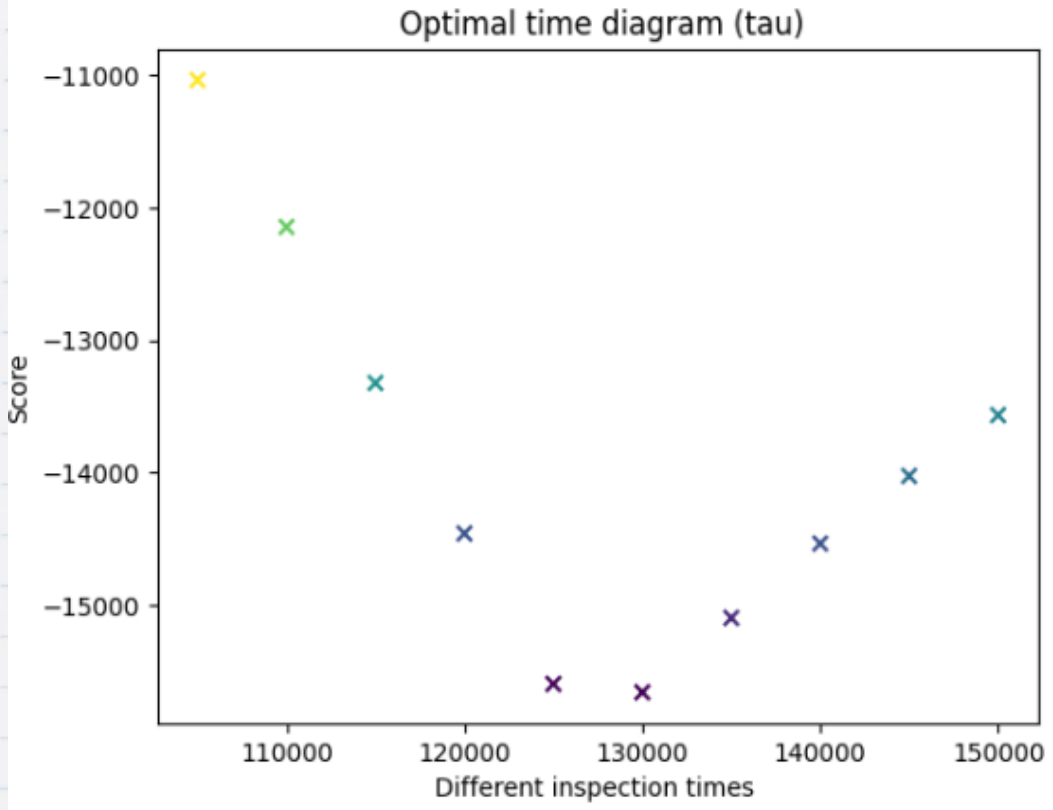
MDP(Marcov decision process):

- استفاده از ماتریس Q-Table

نکته : یافتن اقدام با استفاده از ماتریس بهینه مرحله قبل (بدون بروزرسانی ماتریس)

```python
#fig = plt.figure()
#ax = fig.add_subplot(111)
x = tau_list = [1.05*(10**5),1.10*(10**5),1.15*(10**5),1.20*(10**5),1.25*(10**5)
y =score_tow
plt.scatter(x, y, marker='x', c=y)
plt.ylabel('Score')
plt.xlabel('Different inspection times ')
plt.title(' Optimal time diagram (tau)')
print("this plot is for",t)
plt.show()
```

```
his plot is for 150000.0
```



MDP(Marcov decision process):

- یافتن زمان بهینه بازرسی با ماتریس Q-Table

نکته : بهترین زمان بازرسی جهت کاهش هزینه ها حدود ۱۲۵۰۰۰ واحد زمانی است

پایان