Margot Wagner
A53279875
CSE 250a HW9
Due: 12/10/20

9.1 Effective horizon time

$$\sum_{n \geq t} \gamma^n r_n \qquad \text{where } r_n \in [0,1]$$

$$\leq \sum_{n \geq t}^{\infty} \gamma^n \qquad \text{geometric series } \gamma^t + \gamma^{t+1} + \dots \gamma^\infty$$

$$= \frac{1-\gamma^\infty}{1-\gamma} \gamma^t$$

$$= \frac{1}{1-\gamma} \gamma^t \qquad \gamma < 1 \therefore \gamma^\infty = 0$$

$$= h\gamma^t \qquad h = (1-\gamma)^{-1}$$

$$= he^{\log(\gamma^t)}$$

$$= he^{t\log(\gamma)} \qquad \log\gamma \leq \gamma - 1$$

$$\therefore he^{t\log(\gamma)} \leq he^{t(\gamma-1)} = he^{-t/h} \qquad \text{subst } h \text{ def.}$$

$$\therefore \sum_{n \geq t} \gamma^n r_n \leq he^{-t/h}$$

9.2 3-state, 2-action MDP

$S \in \{1, 2, 3\}$
$a \in \{\downarrow, \uparrow\}$
$\gamma = 3/4$

(a) Policy evaluation

Starting with the Bellman eqtn: (next state)

$$V^\pi(s) = R(s) + \gamma \sum_{s'} P(s' \mid s, \pi(s)) V^\pi(s')$$

$$V^\pi(1) = R(1) + \gamma \sum_{s'} P(s' \mid 1, \pi(1)) V^\pi(s')$$

from lecture 17:

$$V^\pi = (I - \gamma P^\pi)^{-1} R$$

$$P^\pi = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} & & \\ & & \\ & & \end{bmatrix} \end{array}$$

$\rightarrow P(3 \mid 3, \pi(3))$

$$\begin{bmatrix} V^\pi(1) \\ V^\pi(2) \\ V^\pi(3) \end{bmatrix} = \left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \gamma \begin{bmatrix} 1/3 & 2/3 & 0 \\ 2/3 & 1/3 & 0 \\ 0 & 2/3 & 1/3 \end{bmatrix} \right)^{-1} \begin{bmatrix} 24 \\ -6 \\ 12 \end{bmatrix}$$

$$V^\pi = \begin{bmatrix} 48 \\ 24 \\ 32 \end{bmatrix}$$

(b) Greedy policy

$$\pi'(s) = \operatorname{argmax}_a \left[R(s) + \gamma \sum_{s'} P(s' \mid s, a) V^\pi(s') \right]$$

$$= \operatorname{argmax}_a \left[\sum_{s'} P(s' \mid s, a) V^\pi(s') \right]$$

$s = 1$:

$a = \downarrow$ $\quad P(1 \mid 1, \downarrow) V^\pi(1) + P(2 \mid 1, \downarrow) V^\pi(2) + P(3 \mid 1, \downarrow) V^\pi(3)$

$(1/3)(48) + 0(24) + (2/3)(32)$

$= 37.\bar{3}$

$a = \uparrow$ $\Rightarrow 32.0$ calculated similarly

$S = 2$ $a = \downarrow$ $39.\overline{9}$

$\uparrow$ $29.\overline{3}$

$S = 3$ $a = \downarrow$ $26.\overline{6}$

$\uparrow$ $42.\overline{6}$

S	$\pi(s)$	$V^{\pi}(s)$	$\pi'(s)$
1	$\uparrow$	48	$\downarrow$
2	$\downarrow$	24	$\downarrow$
3	$\downarrow$	32	$\uparrow$

9.3 Value function for a random walk

(a) Bellman eqtn

$$V^{\pi}(s) = R(s) + \gamma \sum_{s'} p(s' \mid s, \pi(s)) V^{\pi}(s')$$

$$= R(s) + \gamma \left(\frac{1}{5} V^{\pi}(s) + \frac{3}{5} V^{\pi}(s+1) \right)$$

$$V^{\pi}(s) = s + \frac{2}{5}\gamma V^{\pi}(s) + \frac{3}{5}\gamma V^{\pi}(s+1)$$

(b) $V^{\pi}(s) = as + b$

from part (a)

$$V^{\pi}(s) = s + \frac{2}{5}\gamma V^{\pi}(s) + \frac{3}{5}\gamma V^{\pi}(s+1)$$

subst $V^{\pi}(s) = as + b$

$$as + b = s + \frac{2}{5}\gamma(as + b) + \frac{3}{5}\gamma(a(s+1) + b)$$

$$as + b = s + \frac{2}{5}\gamma as + \frac{2}{5}\gamma b + \frac{3}{5}\gamma as + \frac{3}{5}\gamma a + \frac{3}{5}\gamma b$$

$$as + b = (1 + \gamma a)s + \gamma b + \frac{3}{5}\gamma a$$

$$a = 1 + \gamma a$$

$$a - \gamma a = 1$$

$$a = \frac{1}{1-\gamma}$$

$$b = \gamma b + \frac{3}{5}\gamma a$$

$$b - \gamma b = \frac{3}{5}\gamma a$$

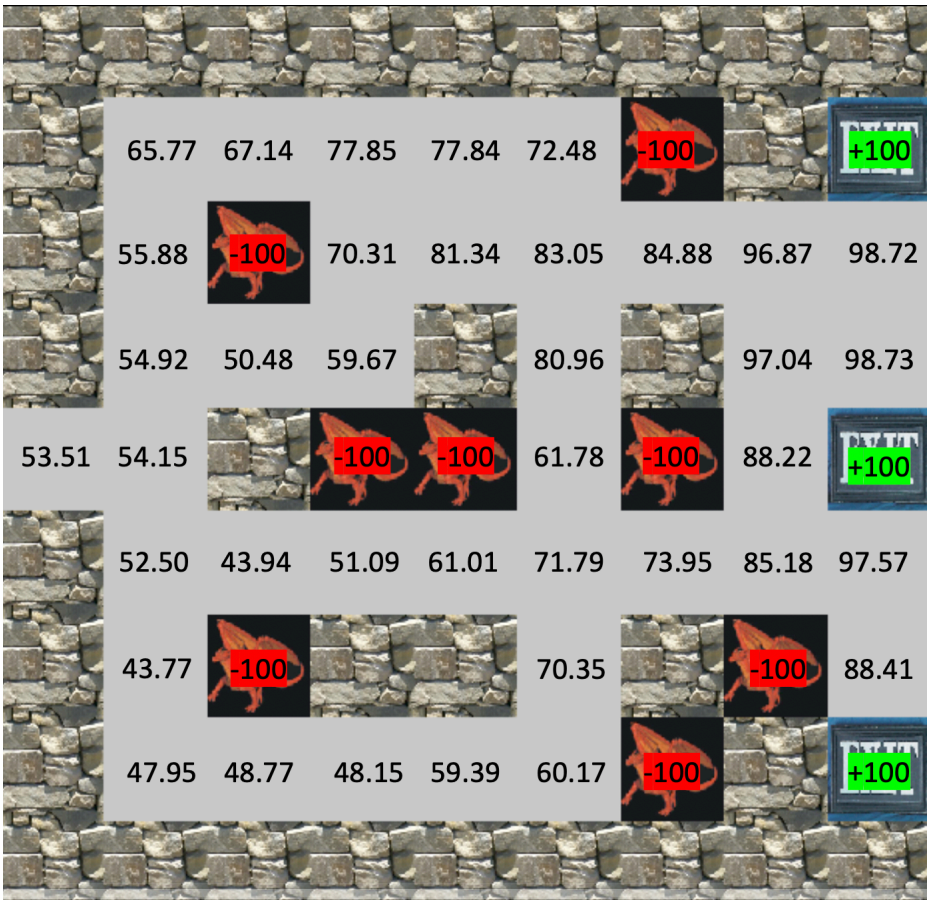
$$b = \frac{\frac{3}{5}\gamma a}{1-\gamma} = \left(\frac{1}{1-\gamma}\right)\left(\frac{3}{5}\gamma\right)\left(\frac{1}{1-\gamma}\right)$$

$$b = \frac{3\gamma}{5(1-\gamma)^2}$$

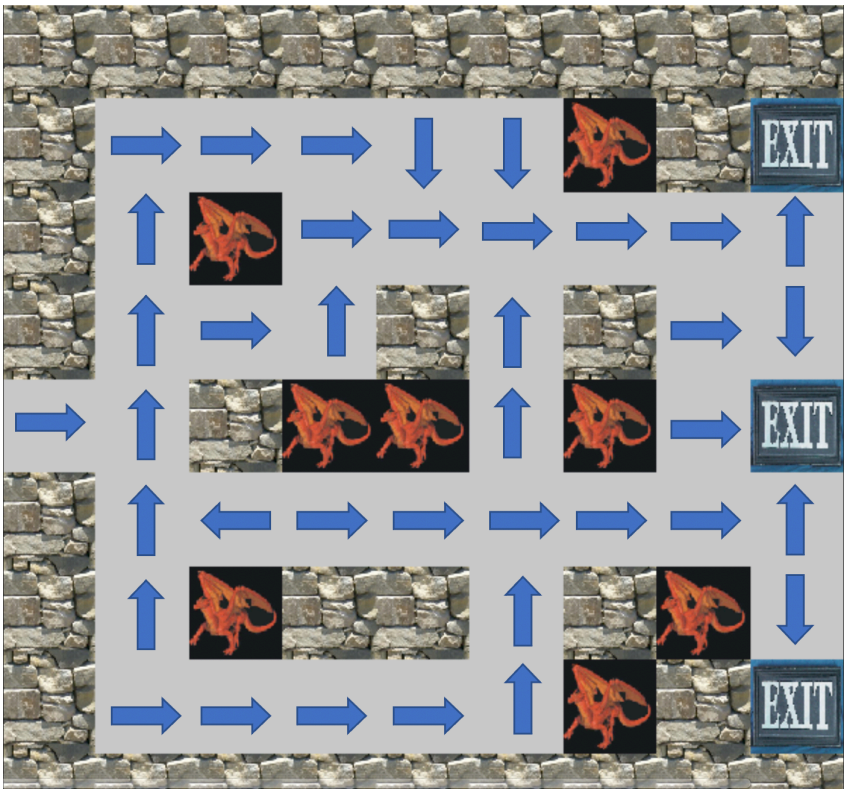
9.4 Policy and value iteration

(a) Policy iteration

(i) Non-zero values:



(ii) Action:



(b) Value iteration

The results are the same as for policy iteration.

(c) Code

9.4

We are given a maze with 81 states (81 squares), 4 actions (up, down, left, right) and discount factor of 0.99.

(a) Policy iteration

In [338]:
```
# constants
gamma = 0.99
r = np.loadtxt('rewards.txt')
p_a1 = np.loadtxt('prob_a1.txt')
s_transitions = p_a1[:,:2].astype('int')
p_a2 = np.loadtxt('prob_a2.txt')
p_a3 = np.loadtxt('prob_a3.txt')
p_a4 = np.loadtxt('prob_a4.txt')
p = np.stack((p_a1[:,2], p_a2[:,2], p_a3[:,2], p_a4[:,2]))
s_start = list(s_transitions[:,0])
N = len(r)
```

In [339]:
```
# 1. Initialize policies randomly
pi_init = [random.randrange(4) for i in range(N)]
```

In [340]:
```
# 2. Evaluate current policy
def v_pi(pi):

    # Create probability matrix
    p_pi = np.zeros((N, N))

    # for each s
    for i in range(N):
        # probability of transition to s'
        s = i+1    # value of state

        # get possible next states given current state
        next_s_i = [j for j, x in enumerate(s_start) if x == s]
        next_s_val = [(s_transitions[j,1]) for j in next_s_i]
        next_s_nonzero_probs = [p[pi[i]][j] for j in next_s_i]

        probs_i = [0]*N
        for v,pr in zip(next_s_val, next_s_nonzero_probs):
            probs_i[v-1] = pr

        p_pi[i, :] = np.array(probs_i) # for all next states


    v = np.dot(np.linalg.inv(np.identity(N) - gamma*p_pi),r)

    return v
```

In [341]:
```
# 3. Update policy
def improve_policy(v):
    opt = []

    # for all states
    for i in range(N):
        s = i+1    # value of state

        if v[i] != 0:
            # get possible next states given current state
            next_s_i = [i for i, x in enumerate(s_start) if x == s]

            # use maximum value across all actions
            max_val = 0
            a_opt = 0
            for a in range(4):
                # value iteration update
                val = r[i] + gamma*sum([p[a][j]*v[s_transitions[j,1] - 1] for j in next_s_i])

                # compare to current max value
                if val > max_val:
                    max_val = val
                    a_opt = a

            # update to maximum value
            opt.append(a_opt)
        else:
            #opt.append('')
            opt.append(0)

    return opt
```

In [342]:
```
pi_curr = pi_init.copy()
pi_next = [0]*N
no_change = pi_next == pi_curr
#for i in range(15):
while not no_change:
    # compute value function
    v_curr = v_pi(pi_curr)

    # improve policy
    pi_next = improve_policy(v_curr)

    # check if change
    no_change = pi_next == pi_curr

    # update pi
    pi_curr = pi_next
```

In [343]:
```
v_curr = v_pi(pi_curr)
np.transpose(np.round(np.array(v_curr),2).reshape((9,9)))
```

Out[343]:
```
array([[   0.  ,    0.  ,    0.  ,    0.  ,    0.  ,    0.  ,
           0.  ,    0.  ],
       [   0.  ,   65.77,   67.14,   77.85,   79.84,   72.48, -100.  ,
           0.  ,  100.  ],
       [   0.  ,   55.88, -100.  ,   70.31,   81.34,   83.05,   84.88,
          96.87,   98.72],
       [   0.  ,   54.92,   50.48,   59.67,    0.  ,   80.96,    0.  ,
          97.04,   98.73],
       [  53.51,   54.15,    0.  , -100.  , -100.  ,   61.78, -100.  ,
          88.22,  100.  ],
       [   0.  ,   52.5 ,   43.94,   51.09,   61.01,   71.79,   73.95,
          85.18,   97.57],
       [   0.  ,   43.77, -100.  ,    0.  ,    0.  ,   70.35,    0.  ,
        -100.  ,   88.41],
       [   0.  ,   47.95,   48.77,   58.15,   59.39,   60.17, -100.  ,
           0.  ,  100.  ],
       [   0.  ,    0.  ,    0.  ,    0.  ,    0.  ,    0.  ,    0.  ,
           0.  ,    0.  ]])
```

In [329]:
```
actions = ['W', 'N', 'E', 'S']
opt_moves = []
for i in range(N):
    if v_curr[i] > 0:
        opt_moves.append(actions[pi_curr[i]])
    else:
        opt_moves.append('')

np.transpose(np.array(opt_moves).reshape((9,9)))
```

Out[329]:
```
array([['', '', '', '', '', '', '', '', ''],
       ['', 'E', 'E', 'E', 'S', 'S', '', '', 'W'],
       ['', 'N', '', 'E', 'E', 'E', 'E', 'E', 'N'],
       ['', 'N', 'E', 'N', '', 'N', '', 'E', 'S'],
       ['E', 'N', '', '', '', 'N', '', 'E', 'W'],
       ['', 'N', 'W', 'E', 'E', 'E', 'E', 'E', 'N'],
       ['', 'N', '', '', '', 'N', '', '', 'S'],
       ['', 'E', 'E', 'E', 'E', 'N', '', '', 'W'],
       ['', '', '', '', '', '', '', '', '']], dtype='<U1')
```

(b) Value iteration

In [331]:
```
# constants
gamma = 0.99
r = np.loadtxt('rewards.txt')
p_a1 = np.loadtxt('prob_a1.txt')
s_transitions = p_a1[:,:2].astype('int')
p_a2 = np.loadtxt('prob_a2.txt')
p_a3 = np.loadtxt('prob_a3.txt')
p_a4 = np.loadtxt('prob_a4.txt')
p = np.stack((p_a1[:,2], p_a2[:,2], p_a3[:,2], p_a4[:,2]))
s_start = list(s_transitions[:,0])
```

In [332]:
```
# 1. Initialize value function for all states (81 functions) to zero
v_init = [0]*81
```

In [333]:
```
def update_v(v):
    v_next = []
    # update all states
    for i in range(81):
        s = i+1    # value of state

        # get possible next states given current state
        next_s_i = [i for i, x in enumerate(s_start) if x == s]

        # use maximum value across all actions
        max_val = 0
        for a in range(4):
            # value iteration update
            val = r[i] + gamma*sum([p[a][j]*v[s_transitions[j,1] - 1] for j in next_s_i])

            # compare to current max value
            if val > max_val:
                max_val = val

        # update to maximum value
        v_next.append(max_val)

    return v_next
```

In [334]:
```
# 2. Iterate until convergence

v_curr = v_init.copy()
diff = 1000000
# 10^-5
while diff > 10**-5:
    v_next = update_v(v_curr)

    diff = sum([i - j for i,j in zip(v_next, v_curr)])

    v_curr = v_next
```

In [344]:
```
np.transpose(np.array(v_curr).reshape((9,9)))
```

Out[344]:
```
array([[   0.        ,    0.        ,    0.        ,    0.        ,
           0.        ,    0.        ,    0.        ,    0.        ,
           0.        ],
       [   0.        ,   65.77308407,   67.13647421,   77.84605   ,
          79.84451583,   72.47511769, -100.        ,    0.        ,
         100.        ],
       [   0.        ,   55.88294346, -100.        ,   70.30818136,
          81.34440225,   83.04847989,   84.88054612,   96.87232244,
          98.71875987],
       [   0.        ,   54.92298013,   50.47656297,   59.66641187,
           0.        ,   80.95826449,    0.        ,   97.04482865,
          98.72729893],
       [  53.50968756,   54.14557214,    0.        , -100.        ,
        -100.        ,   61.77980767, -100.        ,   88.22035599,
         100.        ],
       [   0.        ,   52.50402036,   43.9359876 ,   51.09137525,
          61.00715483,   71.78642614,   73.94661407,   85.18458536,
          97.57257319],
       [   0.        ,   43.77254574, -100.        ,    0.        ,
           0.        ,   70.35142939,    0.        , -100.        ,
          88.40593622],
       [   0.        ,   47.95296148,   48.76871928,   58.14735126,
          59.39003194,   60.1688947 , -100.        ,    0.        ,
         100.        ],
       [   0.        ,    0.        ,    0.        ,    0.        ,
           0.        ,    0.        ,    0.        ,    0.        ,
           0.        ]])
```

In [345]:
```
# 3. Solve for optimal policy
def opt_policy(v):
    opt = []
    actions = ['W', 'N', 'E', 'S']

    # for all states
    for i in range(81):
        s = i+1    # value of state

        if v[i] != 0:
            # get possible next states given current state
            next_s_i = [i for i, x in enumerate(s_start) if x == s]

            # use maximum value across all actions
            max_val = 0
            a_opt = 0
            for a in range(4):
                # value iteration update
                val = r[i] + gamma*sum([p[a][j]*v[s_transitions[j,1] - 1] for j in next_s_i])

                # compare to current max value
                if val > max_val:
                    max_val = val
                    a_opt = actions[a]

            # update to maximum value
            opt.append(a_opt)
        else:
            opt.append('')

    return opt
```

In [347]:
```
np.transpose(np.array(opt_policy(v_curr)).reshape((9,9)))
```

Out[347]:
```
array([['', '', '', '', '', '', '', '', ''],
       ['', 'E', 'E', 'E', 'S', 'S', '', '', 'W'],
       ['', 'N', '', 'E', 'E', 'E', 'E', 'E', 'N'],
       ['', 'N', 'E', 'N', '', 'N', '', 'E', 'S'],
       ['E', 'N', '', '', '', 'N', '', 'E', 'W'],
       ['', 'N', 'W', 'E', 'E', 'E', 'E', 'E', 'N'],
       ['', 'N', '', '', '', 'N', '', '', 'S'],
       ['', 'E', 'E', 'E', 'E', 'N', '', '', 'W'],
       ['', '', '', '', '', '', '', '', '']], dtype='<U1')
```

9.5 Convergence of iterative policy evaluation

$$\Delta_k = \max_s \left| V_k(s) - V^\pi(s) \right|$$

$$V_k(s) = R(s) + \gamma \sum_{s'} P(s' \mid s, \pi(s)) V_{k-1}(s')$$

$$V^\pi(s) = R(s) + \gamma \sum_{s'} P(s' \mid s, \pi(s)) V^\pi(s')$$

$$\begin{aligned}
\Delta_{k+1} &= \max_s \left| V_{k+1}(s) - V^\pi(s) \right| \\
&= \max_s \left| \left[R(s) + \gamma \sum_{s'} P(s' \mid s, a) V_k(s') \right] \right. \\
&\qquad \left. - \left[R(s) + \gamma \sum_{s'} P(s' \mid s, \pi(s)) V^\pi(s') \right] \right| \\
&= \gamma \max_s \left| \sum_{s'} P(s' \mid s, a) V_k(s') - \sum_{s'} P(s' \mid s, a) V^\pi(s') \right| \\
&= \gamma \max_s \left| \sum_{s'} P(s' \mid s, a) \left[V_k(s') - V^\pi(s') \right] \right| \\
&\le \gamma \max_s \left| \sum_{s'} P(s' \mid s, a) \max_{s'} \left| V_k(s') - V^\pi(s') \right| \right| \\
&\le \gamma \max_s \left| \max_{s'} \left| V_k(s') - V^\pi(s') \right| \right| \\
&= \gamma \Delta_k
\end{aligned}$$

$\Delta_{k+1} \le \gamma \Delta_k$

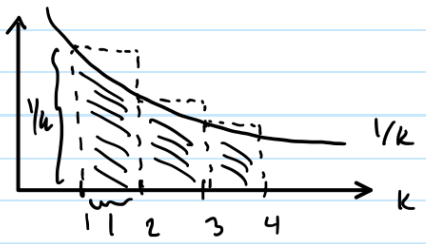
$\therefore$ by induction $\Delta_k \le \gamma^k \Delta_0$

if Δ_0 is bounded, $\lim_{k \to \infty} \Delta_k \to 0$

exponential decay w/ $\gamma < 1$

9.6 Stochastic approximation

(a) (i) $\sum_{k=1}^{\infty} \alpha_k = \sum_{k=1}^{\infty} \frac{1}{k}$



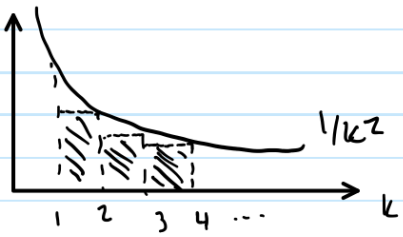
- rectangles are equiv to the sum
- the sum can be approximated initially as the area under the curve.

The area under the curve is the integral by definition:

$$\int_1^{\infty} \frac{1}{x}\,dx = \ln x \Big|_1^{\infty} = \ln \infty - \ln 1 = \infty$$

$$\sum_{k=1}^{\infty} \frac{1}{k} > \int_1^{\infty} \frac{1}{x}\,dx = \infty$$

ii Similarly



$$\sum_{k=1}^{\infty} \frac{1}{k_2} = 1 + \sum_{k=2}^{\infty} \frac{1}{k^2}$$

↳ rectangle areas

$$\sum_{k=2}^{\infty} \frac{1}{k^2} < \int_1^{\infty} \frac{1}{x^2}\,dx$$

$$\sum_{k=1}^{\infty} \frac{1}{k^2} < 1 + \int_1^{\infty} \frac{1}{x^2}\,dx$$

$$\sum_{k=1}^{\infty} \frac{1}{k^2} < 1 + \infty$$

$$< \infty$$

(b) $\mu_0 = 0$

$$\mu_1 = \mu_0 + \frac{1}{1}(x_1 - 0) = x_1$$

$$\mu_2 = \mu_1 + \frac{1}{2}(x_2 - x_1) = x_1 + \frac{1}{2}(x_2 - x_1) = \frac{x_1 + x_2}{2}$$

$$\vdots$$

$$\mu_{k-1} = \frac{x_1 + x_2 + \dots x_{k-1}}{k-1}$$

$$\mu_k = \mu_{k-1} + \alpha_k (x_k - \mu_{k-1})$$

$$= \frac{x_1 + x_2 + \dots x_{k-1}}{k-1} + \frac{1}{k}\left(x_k - \frac{x_1 + x_2 + \dots x_{k-1}}{k-1}\right)$$

$$= \frac{x_1 + x_2 + \dots x_{k-1}}{k-1} + \frac{1}{k}\left(\frac{(k-1)(x_k) - (x_1 + x_2 + \dots + x_{k-1})}{k-1}\right)$$

$$= \frac{k(x_1 + x_2 + \dots + x_{k-1}) + (k-1)x_k - (x_1 + x_2 + \dots + x_{k-1})}{k(k-1)}$$

$$= \frac{k(x_1 + \dots x_{k-1}) + kx_k - x_k - (x_1 \dots x_{k-1})}{k(k-1)}$$

$$= \frac{(k-1)(x_1 + x_2 + \dots x_k)}{k(k-1)}$$

$$\mu_k = \frac{(x_1 + x_2 + \dots + x_k)}{k}$$