1. $V^*(s) = \max_{a \in A} \{ R(s,a) + 0 \cdot \Sigma \cdots \}$

$$= \max_{a \in A} \left\{ \int_{-\infty}^{\infty} P(s,a,s') R_T(s,a,s') \, ds' \right\}$$

$$= \max_{a \in A} \left\{ \int_{-\infty}^{\infty} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{s'-s}{\sigma}\right)^2} \left(-e^{as'}\right) ds' \right\}$$

$$= \max_{a \in A} \left\{ -e^{\frac{a(2s+\sigma^2 a)}{2}} \int_{-\infty}^{\infty} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{s'-s-\sigma^2 a}{\sigma}\right)^2} ds' \right\}$$

$$= \max_{a \in A} \left\{ -e^{a(2s+\sigma^2 a)/2} \right\}$$

Minimize $a(2s+\sigma^2 a)/2$

$$\Rightarrow \frac{\partial}{\partial a}\left(as + \sigma^2 a^2\right) = 0$$

$$s = -\sigma^2 a$$

$$a = -s/\sigma^2$$

cost is $e^{\frac{-ss'}{\sigma^2}}$

2. $V_0(s_1)=10,\ V_0(s_2)=1,\ V_0(s_3)=0$

$q_1(s_1,a_1)=R(s_1,a_1)+P(s_1,a_1,s_1)V(s_1)$
$+P(s_1,a_1,s_2)V(s_2)$
$+P(s_1,a_1,s_3)V(s_3)$
$=8+0.2\cdot 10+0.6\cdot 1+0=10.6$

$q_1(s_1,a_2)=R(s_1,a_2)+P(s_1,a_2,s_1)V(s_1)+P(s_1,a_2,s_2)V(s_2)$
$+P(s_1,a_2,s_3)V(s_3)$
$=10+0.1\cdot 10+0.2\cdot 1+0=11.2$

$q_1(s_2,a_1)=R(s_2,a_1)+P(s_2,a_1,s_1)V(s_1)+P(s_2,a_1,s_2)V(s_2)$
$=1+0.3\cdot 10+0.3\cdot 1=4.3$

$q_1(s_2,a_2)=R(s_2,a_2)+P(s_2,a_2,s_1)V(s_1)+P(s_2,a_2,s_2)V(s_2)$
$=-1+0.5\cdot 10+0.3\cdot 1=4.3$

$V_1(s_1)=\max_{a\in A} q_1(s_1,a)=q_1(s_1,a_2)=11.2$

$V_1(s_2)=\max_{a\in A} q_1(s_2,a)=q_1(s_2,a_1)=4.3\quad(=q_1(s_2,a_2))$

$V_1(s_3)=0$

$\pi_0(s_1)=a_2\quad \pi_1(s_2)=a_1$

$q_2(s_1,a_1)=R(s_1,a_1)+P(s_1,a_1,s_1)V(s_1)+P(s_1,a_1,s_2)V(s_2)$
$=8+0.2\cdot 11.2+0.6\cdot 4.3=12.82$

$q_2(s_1,a_2)=R(s_1,a_2)+P(s_1,a_2,s_1)V(s_1)+P(s_1,a_2,s_2)V(s_2)$
$=10+0.1\cdot 11.2+0.2\cdot 4.3=11.98$

$q_2(s_2,a_1)=R(s_2,a_1)+P(s_2,a_1,s_1)V(s_1)+P(s_2,a_1,s_2)V(s_2)$
$=1+0.3\cdot 11.2+0.3\cdot 4.3=5.65$

$q_2(s_2,a_2)=R(s_2,a_2)+P(s_2,a_2,s_1)V(s_1)+P(s_2,a_2,s_2)V(s_2)$
$=-1+0.5\cdot 11.2+0.3\cdot 4.3=5.89$

$V_2(s_1)=\max_{a\in A} q_2(s_1,a)=q_2(s_1,a_1)=12.82$

$V_2(s_2)=\max_{a\in A} q_2(s_2,a)=q_2(s_2,a_2)=5.89$

$\pi_2(s_1)=a_1\quad \pi_2(s_2)=a_2$

In order for π_2 to be optimal ($\pi_K=\pi_2\ \forall K\geq 2$), we need

$q_K(s_1,a_1)\geq q_K(s_1,a_2)$

$\Rightarrow 8+0.2V_{K-1}(s_1)+0.6V_{K-1}(s_2)\geq 10+0.1V_{K-1}(s_1)+0.2V_{K-1}(s_2)\quad (1)$

$q_K(s_2,a_2)\geq q_K(s_2,a_1)\Rightarrow -1+0.5V_{K-1}(s_1)+0.3V_{K-1}(s_2)\geq 1+0.3V_{K-1}(s_1)+0.3V_{K-1}(s_2)$

$\Rightarrow 0.1V_{K-1}(s_1)+0.4V_{K-1}(s_2)\geq 2,\quad 0.2V_{K-1}(s_1)\geq 2$

For $V_{K-1}=V_2$, these inequalities hold. They will continue to hold for successive iterations because V_K is nondecreasing, so the inequalities will continue to hold.

3.

$S = \{1, \dots, n\} \times \{E, N\}$

index of job offered, employed or not employed when offered

$A = \{A, R\}$, accept/reject offer when not employed (N)

$A = \{A\}$, accept offer when already employed (E)

$$P((i,j), a, (i',j')) = \begin{cases} 1-\alpha & \text{if } i'=i,\ j'=E,\ j \in \{E,N\},\ i,i' \in \{1,\dots,n\} \\ \alpha \cdot p_{i'} & \text{if } a=A,\ j'=N,\ i' \in \{1,\dots,n\} \\ p_{i'} & \text{if } a=R,\ j=N,\ j'=N,\ i,i' \in \{1,\dots,n\} \\ 0 & \text{o.w.} \end{cases}$$

$$R((i,j), a) = \begin{cases} \log w_i & \text{if } a=A \\ \log w_0 & \text{if } a=R \end{cases}$$

$$V^*((i,j)) = \max_{a \in A} \left\{ R((i,j), a) + \gamma \cdot \sum_{s' \in S} P((i,j), a, s') V^*(s) \right\}$$

$$V^*((i,E)) = \log w_i + \gamma \cdot (1-\alpha) V^*((i,E)) + \gamma \cdot \alpha \sum_{i' \in \{1,\dots,n\}} p_{i'} V^*((i',N))$$

$$V^*((i,N)) = \max_{a \in A} \left\{ R((i,N), a) + I_{\{a=A\}} \gamma \cdot \left((1-\alpha) V^*((i,E)) + \alpha \sum_{i' \in \{1,\dots,n\}} p_{i'} V^*((i',N)) \right) + I_{\{a=R\}} \gamma \sum_{i'} p_{i'} V^*((i',N)) \right\}$$

$$V^*((i,N)) = \max \left\{ \log w_0 + \gamma \sum_{i'} p_{i'} V^*((i',N)),\ V^*((i,E)) \right\}$$

In [6]:
```
import numpy as np
from typing import Mapping, Sequence, Tuple
from dataclasses import dataclass
```

In [65]:
```
@dataclass
class OptimalJobs():
    value_function: Mapping[Tuple[int, str], float]
    probabilities: Sequence[float]
    wages: Sequence[float]
    gamma: float
    alpha: float
    def transition_probability(self, in_state: Tuple[int, str], out_state: Tuple[int, st
        in_job = in_state[0]
        in_employ = in_state[1]
        out_job = out_state[0]
        out_employ = out_state[1]
        if (action == 'R') & (in_employ == 'N') & (out_employ == 'N'):
            return self.probabilities[out_job - 1]
        if (action == 'A') & (out_employ == 'N'):
            return self.alpha * self.probabilities[out_job - 1]
        if (in_job == out_job) & (out_employ == 'E'):
            return 1 - self.alpha
        else:
            return 0
    def expected_reward(self, in_state: Tuple[int, str], action: 'str') -> float:
        if action == 'A':
            return self.wages[in_state[0]]
        elif action == 'R':
            return self.wages[0]

    def vf_if_employ(self, in_state: Tuple[int, str]) -> float:
        value = np.log(self.wages[in_state[0]]) + self.gamma * (1 - self.alpha) \
                * self.value_function[tuple([in_state[0], 'E'])]
        other_vals = np.array([self.value_function[tuple([i, 'N'])] for i in range(1, le
        value += np.sum(other_vals * self.probabilities) * self.gamma * self.alpha
        return value

    def vf_if_notemploy(self, in_state: Tuple[int, str]) -> float:
        value = np.log(self.wages[0]) + self.gamma \
                * np.sum(np.array([self.value_function[tuple([i, 'N'])] for i in range(1
                * self.probabilities)
        value = max(value, self.vf_if_employ(in_state))
        return value

    def bellman_operator(self):
        vf_copy = self.value_function.copy()
        for in_state in vf_copy.keys():
            if in_state[1] == 'E':
                val = self.vf_if_employ(in_state)
            elif in_state[1] == 'N':
                val = self.vf_if_notemploy(in_state)
            vf_copy[in_state] = val
        self.value_function = vf_copy

    def get_actions(self) -> Mapping[Tuple[int, str], str]:
        vf_copy = self.value_function.copy()
        for in_state in vf_copy.keys():
            if in_state[1] == 'E':
                vf_copy[in_state] = 'A'
            if in_state[1] == 'N':
                if self.vf_if_employ(in_state) >= self.vf_if_notemploy(in_state):
                    vf_copy[in_state] = 'A'
```

```
                else:
                    vf_copy[in_state] = 'R'
        return vf_copy

    def compute_value_function(self, tolerance: float = 1e-4) -> Mapping[Tuple[int, str]
        keys = self.value_function.keys()
        diff = float('inf')
        while diff > tolerance:
            old_vals = np.array([self.value_function[key] for key in keys])
            self.bellman_operator()
            new_vals = np.array([self.value_function[key] for key in keys])
            diff = np.max(np.abs(new_vals - old_vals))
        return self.value_function
```

In [66]:
```
n = 10
rands = np.random.rand(10)
probs = rands / rands.sum()
wages = np.random.rand(11) * 100
dict_keys = [tuple([i, c]) for c in ['E', 'N'] for i in range(1, n + 1)]
value_function = dict(zip(dict_keys, np.zeros(n * 2)))
gamma = 0.5
alpha = 0.2
```

In [67]:
```
optim = OptimalJobs(value_function=value_function, probabilities=probs, wages=wages, gam
```

In [68]:
```
optim.compute_value_function()
```

Out[68]:
```
{(1, 'E'): 8.965454174298525,
 (2, 'E'): 8.728637113705796,
 (3, 'E'): 8.858567476725499,
 (4, 'E'): 6.771465687378746,
 (5, 'E'): 8.731514108195988,
 (6, 'E'): 7.187558652229709,
 (7, 'E'): 7.944154937384902,
 (8, 'E'): 7.486429367781601,
 (9, 'E'): 7.158429294244973,
 (10, 'E'): 6.974224725439922,
 (1, 'N'): 8.965454174298525,
 (2, 'N'): 8.728637113705796,
 (3, 'N'): 8.858567476725499,
 (4, 'N'): 8.401314289192019,
 (5, 'N'): 8.731514108195988,
 (6, 'N'): 8.401314289192019,
 (7, 'N'): 8.401314289192019,
 (8, 'N'): 8.401314289192019,
 (9, 'N'): 8.401314289192019,
 (10, 'N'): 8.401314289192019}
```

In [69]:
```
optim.get_actions()
```

Out[69]:
```
{(1, 'E'): 'A',
 (2, 'E'): 'A',
 (3, 'E'): 'A',
 (4, 'E'): 'A',
 (5, 'E'): 'A',
 (6, 'E'): 'A',
 (7, 'E'): 'A',
 (8, 'E'): 'A',
 (9, 'E'): 'A',
 (10, 'E'): 'A',
 (1, 'N'): 'A',
 (2, 'N'): 'A',
 (3, 'N'): 'A',
 (4, 'N'): 'R',
 (5, 'N'): 'A',
 (6, 'N'): 'R',
 (7, 'N'): 'R',
 (8, 'N'): 'R',
 (9, 'N'): 'R',
 (10, 'N'): 'R'}
```