William Bidle

Final Question 2

- For each of the 10 states, what is the optimal utility (long term expected discounted value) available in that state (i.e., U $*$ (state))?

Given a state *s*, taking action *a* in our action set *A(s)* will move us to state *s'* with probability $p^a_{s,s'}$. In our case the states are New – Dead, and our action set is to either use if we are in states New – Used8 or replace if we are in states Used1 – Dead. Performing an action also has a reward associated to it, $r_{s,a}$, which is positive is we use, and negative if we replace. Therefore, given a β, we can iteratively calculate an estimate for the optimal utility of the sequence by finding updating the utility of each step based on choosing the action that will provide us the greatest utility, factoring in future situations. This iterative process was calculated by using the following equation:

$$U^*_{k+1}(s) = \max_{a \in A(s)} \left[r_{s,a} + \beta \sum_{s'} p^a_{s,s'} \, U^*_k(s') \right]$$

As seen, I am updating the optimal utility of a step *s*, $U^*_{k+1}(s)$, with the maximum argument of what the expected reward is for that action, plus the sum of all expected future utility of taking that action (in this case the sum has only one term if the machine is replaced, and has two terms if the machine is used, since we either move on or stay put). This equation also reveals to us the relevance of β, as the higher β is, the more the future has to be taken into account, potentially changing our decision process. Since β is relatively high from the start, I expect that the optimal policy I will find will indicate to replace sooner rather than later.

Using the information provided in the problem, I first found when the utility of the system would converge. I did this by running my algorithm over a large set of restarts, ranging from 0 to 100, and plotting the results of each step (see Figure 1). From my results, I found that the optimal number of restarts with β = 0.9 was around 50.

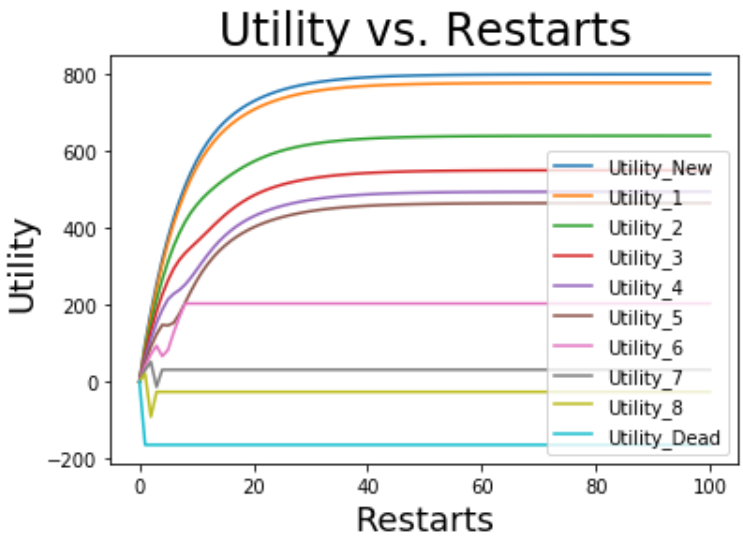


Figure 1. Convergence of Utility

I then ran my algorithm for this many restart and found the optimal utilities for each step (see Figure 2). These results tell us the long term expected value we would get for progressing to that step. It can be seen that the long term expected utility of each step slowly decreases as we keep using the machine (as expected).

```
Running through optimal times for beta = 0.9:

UTILITY OF STEP 0 : 796.0520451309613
UTILITY OF STEP 1 : 773.6913341143916
UTILITY OF STEP 2 : 636.3358629764357
UTILITY OF STEP 3 : 546.8342214100409
UTILITY OF STEP 4 : 491.0321809818268
UTILITY OF STEP 5 : 461.4468406178652
UTILITY OF STEP 6 : 202.552442804662
UTILITY OF STEP 7 : 30.452190000000087
UTILITY OF STEP 8 : -27.218999999999966
UTILITY OF STEP 9 : -165.0
```

Figure 2. Optimal Utility

- What is the optimal policy that gives you this optimal utility - i.e., in each state, what is the best action to take in that state?

To find the optimal policy of each step, I kept track of the moves made at each instance over the 50 steps; either use or replace (see Figure 3). We can see that in this situation we would use our machine for the steps New – Used4 and replace it at Used5. The steps Used6 – Dead would never need to be considered since we will always replace at Used4[1]. To be sure that this policy is optimal for the number of replacements used, I replicated the data with 99 replacements and still found the results to be the same (see Figure 4). It can be seen that neither the uses nor

[1] In the miraculous case we end up in any of states Used6 – Dead, we would obviously replace no matter what.

replacements for Used6 – Dead increase, indicating that we never even get to them. Only the replacements of Used5 increases, which confirms that the optimal policy would be to replace at Used5 for $\beta = 0.9$.

```
NUM TIMES REPLACED: [ 0.  0.  0.  0.  0. 42.  4.  1.  2.  1.]
NUM TIMES USED:     [50. 50. 50. 50. 50.  8.  4.  3.  1.  0.]
```

Figure 3. Optimal Policy

```
Just to make sure it doesn't change for larger replacements...
NUM TIMES REPLACED: [ 0.  0.  0.  0.  0. 91.  4.  1.  2.  1.]
NUM TIMES USED:     [99. 99. 99. 99. 99.  8.  4.  3.  1.  0.]
```

Figure 4. Optimal Policy for Larger Replacements

- For different values of β (such that $0 < \beta < 1$), the utility or value of being in certain states will change. However, the optimal policy may not. Compare the optimal policy for $\beta = 0.1$, 0.3, 0.5, 0.7, 0.9, 0.99, etc. Is there a policy that is optimal for all sufficiently large β? Does this policy make sense? Explain.

This time I did the same process for calculating the optimal policy, but changed the value of β. I used the recommended values 0.1, 0.3, 0.5, 0.7, 0.9, and 0.99, as well as 0.95 and 1[2] to add some more 'high value' β's (see Figure 5). Each sequence used the optimal number of restarts calculated above, and we can see varying results for the optimal policy depending on the value of β. For $0 < \beta \leq 0.7$, we can see that the optimal policy is always to use the machine until it is Dead, and only then will we replace it. This does seem to make sense, as for a lower β, the future is not as important, and we will keep using the machine until it breaks. For a 'sufficiently large' β, (i.e. $0.7 < \beta < 1$), we find that the policy starts to change and force us to make earlier replacements of our machine. Again, this agrees with intuition, as we would want to replace the machine sooner if we knew the future mattered more. If, for example, we needed this machine to perform tasks for us in a time crunch, we would need it to be as efficient as possible in the time we are given, so we would find ourselves replacing it more frequently than if we had less of a time constraint. It is hard to define an exact optimal policy to follow for all 'sufficiently large' β given how small deviations in β can cause the policy to change pretty immediately. For example, moving from $\beta = 0.9$ to $\beta = 0.95$ changes the policy from replace at Used5 to replace at Used4. Moving further from $\beta = 0.95$ to $\beta = 0.99$ changes the policy again from replace at Used4 to replace at Used3. Therefore, I would conclude there isn't an optimal policy that perfectly encapsulates all of the 'sufficiently large' β's. If I had to pick one, I would potentially argue to replace at Used4, since it is the median of the large β's, although again using this policy for $\beta = 0.9$, for instance, would cause us to be replacing early and not getting the most bang for our buck, to say.

[2] Even though we are considering $0 < \beta < 1$, I included this to see what the upper limit looks like.

```
For beta = 0.1
NUM TIMES REPLACED: [ 0.  0.  0.  0.  0.  0.  0.  0.  0. 50.]
NUM TIMES USED:     [50. 50. 50. 50. 50. 50. 50. 50. 50.  0.]

For beta = 0.3
NUM TIMES REPLACED: [ 0.  0.  0.  0.  0.  0.  0.  0.  0. 50.]
NUM TIMES USED:     [50. 50. 50. 50. 50. 50. 50. 50. 50.  0.]

For beta = 0.5
NUM TIMES REPLACED: [ 0.  0.  0.  0.  0.  0.  0.  0.  0. 50.]
NUM TIMES USED:     [50. 50. 50. 50. 50. 50. 50. 50. 50.  0.]

For beta = 0.7
NUM TIMES REPLACED: [ 0.  0.  0.  0.  0.  0.  0.  0.  0. 50.]
NUM TIMES USED:     [50. 50. 50. 50. 50. 50. 50. 50. 50.  0.]

------------------------Large Betas-------------------------
For beta = 0.9
NUM TIMES REPLACED: [ 0.  0.  0.  0.  0. 42.  4.  1.  2.  1.]
NUM TIMES USED:     [50. 50. 50. 50. 50.  8.  4.  3.  1.  0.]

For beta = 0.95
NUM TIMES REPLACED: [ 0.  0.  0.  0. 42.  3.  1.  2.  1.  1.]
NUM TIMES USED:     [50. 50. 50. 50.  8.  5.  4.  2.  1.  0.]

For beta = 0.99
NUM TIMES REPLACED: [ 0.  0.  0. 40.  4.  2.  1.  1.  1.  1.]
NUM TIMES USED:     [50. 50. 50. 10.  6.  4.  3.  2.  1.  0.]

For beta = 1
NUM TIMES REPLACED: [ 0.  0.  0. 41.  3.  2.  1.  1.  1.  1.]
NUM TIMES USED:     [50. 50. 50.  9.  6.  4.  3.  2.  1.  0.]
```

Figure 5. Optimal Policy for a Range of Beta's

In [152]:
```
import numpy as np
import matplotlib.pyplot as plt
```

Define the Algorithm:

In [153]:
```
# Formula's based off of the 'ValueIteration' Notes
def MDP(sequence, index, beta, numReplaced, numUsed):

    #variable to check if we replaced the robot
    replaced = False

    ### the NEW state ###
    if index == 0:
        # USE
        reward = 100

        # find the utility of using the machine
        use_util = reward + beta*sequence[index + 1]

        arg_max = use_util

        numUsed[index] += 1

        sequence[index] = arg_max

    ### the DEAD state ###
    elif index == 9:

        # REPLACE
        reward = -255

        # find the utility of replacing the machine
        replace_util = reward + beta*sequence[0]

        arg_max = replace_util

        numReplaced[index] += 1

        sequence[index] = arg_max

        # keep track that we replaced the machine
        replaced = True

    ### the USE_1 - USE_8 states ###
    else:

        # USE
        reward = 100 - 10*index
        prob_transition = 0.1*index
        prob_stay = 1 - prob_transition

        # find the utility of using the machine
        use_util = reward + beta*prob_transition*sequence[index + 1] + b
eta*prob_stay*sequence[index]

        # REPLACE
        reward = -255

        # find the utility of replacing the machine
        replace_util = reward + beta*sequence[0]
```

```
        # get the argmax and choose this
        arg_max = max(use_util,replace_util)
        sequence[index] = arg_max

        # if we replaced we need to keep track
        if arg_max == replace_util:
            numReplaced[index] += 1
            replaced = True

        # if we used we need to keep track
        else:
            numUsed[index] += 1

    return replaced, numReplaced, numUsed

# lets us set the beta and number of runs through
def main(beta, numRuns):
    numRestarts = 0
    sequence = np.zeros(10)

    # these keep track of decisions made for each step
    numReplaced = np.zeros(10)
    numUsed = np.zeros(10)

    # run until we have restarted the number of times specified
    while numRestarts < numRuns:
        # step through the sequence
        for index in range(10):
            replaced, numReplaced, numUsed = MDP(sequence, index, beta,
numReplaced, numUsed)

            # if we replaced the machine, add 1 to number of restarts an
d break the for loop
            if replaced:
                numRestarts += 1
                break

    return sequence, numReplaced, numUsed
```

Analysis:

In [154]:
```
print("Running through once for beta = 9:")
print()

sequence, numReplaced, numUsed = main(0.9, 1)
for item in sequence:
    print("UTILITY OF STEP", np.where(sequence == item)[0][0], ":",item)

print()

print("NUM TIMES REPLACED:", numReplaced)
print("NUM TIMES USED:    ", numUsed)

print()
print("We don't have enough information to determine optimal policy for
 each step yet... so let's find the convergence")
```

```
Running through once for beta = 9:

UTILITY OF STEP 0 : 100.0
UTILITY OF STEP 1 : 90.0
UTILITY OF STEP 2 : 80.0
UTILITY OF STEP 3 : 70.0
UTILITY OF STEP 4 : 60.0
UTILITY OF STEP 5 : 50.0
UTILITY OF STEP 6 : 40.0
UTILITY OF STEP 7 : 30.0
UTILITY OF STEP 8 : 20.0
UTILITY OF STEP 9 : -165.0

NUM TIMES REPLACED: [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
NUM TIMES USED:     [1. 1. 1. 1. 1. 1. 1. 1. 1. 0.]

We don't have enough information to determine optimal policy for each s
tep yet... so let's find the convergence
```

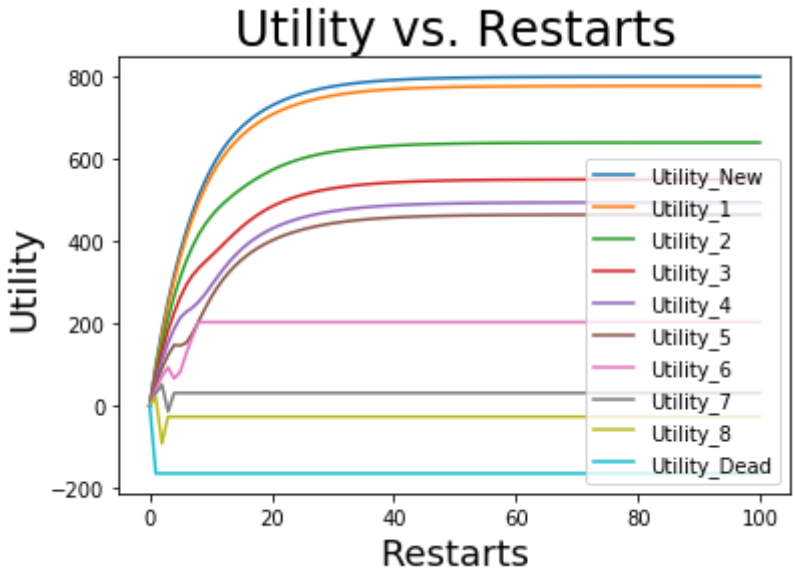
In [155]:

```
# Find Convergence for a fixed beta

x = np.linspace(0,100,101)
Utility_0,Utility_1,Utility_2,Utility_3,Utility_4,Utility_5,Utility_6,Ut
ility_7,Utility_8,Utility_9 = [],[],[],[],[],[],[],[],[],[]
for item in x:
    sequence = main(0.9, item)[0]
    Utility_0.append(sequence[0])
    Utility_1.append(sequence[1])
    Utility_2.append(sequence[2])
    Utility_3.append(sequence[3])
    Utility_4.append(sequence[4])
    Utility_5.append(sequence[5])
    Utility_6.append(sequence[6])
    Utility_7.append(sequence[7])
    Utility_8.append(sequence[8])
    Utility_9.append(sequence[9])

plt.plot(x, Utility_0, label = 'Utility_New')
plt.plot(x, Utility_1, label = 'Utility_1')
plt.plot(x, Utility_2, label = 'Utility_2')
plt.plot(x, Utility_3, label = 'Utility_3')
plt.plot(x, Utility_4, label = 'Utility_4')
plt.plot(x, Utility_5, label = 'Utility_5')
plt.plot(x, Utility_6, label = 'Utility_6')
plt.plot(x, Utility_7, label = 'Utility_7')
plt.plot(x, Utility_8, label = 'Utility_8')
plt.plot(x, Utility_9, label = 'Utility_Dead')
plt.xlabel('Restarts',fontsize=18)
plt.ylabel('Utility',fontsize=18)
plt.title("Utility vs. Restarts", fontsize = 24)
plt.legend(loc = 'lower right')
plt.show()

print("Our data appears to hit the limit around 50, so let's set that as
the optimal number of runs")
```



Our data appears to hit the limit around 50, so let's set that as the o
ptimal number of runs

In [156]:
```
print("Running through optimal times for beta = 0.9:")
print()

sequence, numReplaced, numUsed = main(0.9, 50)
for item in sequence:
    print("UTILITY OF STEP", np.where(sequence == item)[0][0], ":",item)

print()

print("NUM TIMES REPLACED:", numReplaced)
print("NUM TIMES USED:    ", numUsed)

print()
print("We can determine optimal policy based on comparing the number of
 times a spot was replaced or used")

print()
print("Just to make sure it doesn't change for larger replacements...")
sequence, numReplaced, numUsed = main(0.9, 99)

print("NUM TIMES REPLACED:", numReplaced)
print("NUM TIMES USED:    ", numUsed)
```

```
Running through optimal times for beta = 0.9:

UTILITY OF STEP 0 : 796.0520451309613
UTILITY OF STEP 1 : 773.6913341143916
UTILITY OF STEP 2 : 636.3358629764357
UTILITY OF STEP 3 : 546.8342214100409
UTILITY OF STEP 4 : 491.0321809818268
UTILITY OF STEP 5 : 461.4468406178652
UTILITY OF STEP 6 : 202.552442804662
UTILITY OF STEP 7 : 30.452190000000087
UTILITY OF STEP 8 : -27.218999999999966
UTILITY OF STEP 9 : -165.0

NUM TIMES REPLACED: [ 0.  0.  0.  0.  0. 42.  4.  1.  2.  1.]
NUM TIMES USED:     [50. 50. 50. 50. 50.  8.  4.  3.  1.  0.]

We can determine optimal policy based on comparing the number of times
a spot was replaced or used

Just to make sure it doesn't change for larger replacements...
NUM TIMES REPLACED: [ 0.  0.  0.  0.  0. 91.  4.  1.  2.  1.]
NUM TIMES USED:     [99. 99. 99. 99. 99.  8.  4.  3.  1.  0.]
```

In [157]:
```
# different betas

beta_list = [.1,.3,.5,.7,.9,.95,.99, 1]
for item in beta_list:
    if item == 0.9:
        print('-------------------------Large Betas----------------------
----')
    sequence, numReplaced, numUsed = main(item, 50)
    print("For beta =", item)
    print("NUM TIMES REPLACED:", numReplaced)
    print("NUM TIMES USED:    ", numUsed)
    print()
```

```
For beta = 0.1
NUM TIMES REPLACED: [ 0.  0.  0.  0.  0.  0.  0.  0.  0. 50.]
NUM TIMES USED:     [50. 50. 50. 50. 50. 50. 50. 50. 50.  0.]

For beta = 0.3
NUM TIMES REPLACED: [ 0.  0.  0.  0.  0.  0.  0.  0.  0. 50.]
NUM TIMES USED:     [50. 50. 50. 50. 50. 50. 50. 50. 50.  0.]

For beta = 0.5
NUM TIMES REPLACED: [ 0.  0.  0.  0.  0.  0.  0.  0.  0. 50.]
NUM TIMES USED:     [50. 50. 50. 50. 50. 50. 50. 50. 50.  0.]

For beta = 0.7
NUM TIMES REPLACED: [ 0.  0.  0.  0.  0.  0.  0.  0.  0. 50.]
NUM TIMES USED:     [50. 50. 50. 50. 50. 50. 50. 50. 50.  0.]

-------------------------Large Betas----------------------
For beta = 0.9
NUM TIMES REPLACED: [ 0.  0.  0.  0.  0. 42.  4.  1.  2.  1.]
NUM TIMES USED:     [50. 50. 50. 50. 50.  8.  4.  3.  1.  0.]

For beta = 0.95
NUM TIMES REPLACED: [ 0.  0.  0.  0. 42.  3.  1.  2.  1.  1.]
NUM TIMES USED:     [50. 50. 50. 50.  8.  5.  4.  2.  1.  0.]

For beta = 0.99
NUM TIMES REPLACED: [ 0.  0.  0. 40.  4.  2.  1.  1.  1.  1.]
NUM TIMES USED:     [50. 50. 50. 10.  6.  4.  3.  2.  1.  0.]

For beta = 1
NUM TIMES REPLACED: [ 0.  0.  0. 41.  3.  2.  1.  1.  1.  1.]
NUM TIMES USED:     [50. 50. 50.  9.  6.  4.  3.  2.  1.  0.]
```

In [ ]: