TSEAT Algorithm

Parameter Weight matrix, P[m][m]

1. Algorithm
   a. User answer the pairwise comparisons between two parameters (m)
   b. The options are: equally, moderately, greatly, extremely,
      i. respect to weights are: 1.0, 1.2,1.5,2.0
   c. Build up a consistent matrix. Transform matrix using the inverse function
   d. Normalize elements in P dividing by the summation of column

1	f(a)	f(a)f(b)	f(a)f(b)f(c)	f(a)f(b)f(c)f(d)
$\frac{1}{f(a)}$	1	f(b)	f(b)f(c)	f(b)f(c)f(d)
$\frac{1}{f(a)f(b)}$	$\frac{1}{f(b)}$	1	f(c)	f(c)(d)
$\frac{1}{f(a)f(b)f(c)}$	$\frac{1}{f(b)f(c)}$	$\frac{1}{f(c)}$	1	f(d)
$\frac{1}{\mathrm{f(a)f(b)f(c)f(d}}$	$\frac{1}{f(b)f(c)f(d)}$	$\frac{1}{f(c)f(d)}$	$\frac{1}{f(d)}$	1

2. Pseudo code
   a. For i in range (m): comparisons[i] = a, b, …, m-1
   b. For i in range (m):
      i. If comparisons [i] == equally
         P[i][i]=1.0; P[i][i+1]=1.0; P[i+1][i]=1.0/P[i][i+1];
      ii. If comparisons [i] == moderately
         P[i][i]=1.0; P[i][i+1]=1.2; P[i+1][i]=1.0/P[i][i+1];
      iii. If comparisons [i] == greatly
         P[i][i]=1.0; P[i][i+1]=1.5; P[i+1][i]=1.0/P[i][i+1];
      iv. If comparisons [i] == extremely
         P[i][i]=1.0; P[i][i+1]=2.0; P[i+1][i]=1.0/P[i][i+1];
   c. for(int i=0; i<m-2; i++)

```
for(int j=i+2; j<m; j++)
        P[i][j]=P[i][j-1]*P[j-1][j];
        P[j][i]=1.0/P[i][j];
```

   d. For i in range (m):

      sum1stColP+= P[i][0]

      For i in range (m):

      parameterWeights[i] = W[i] = P[i][0] / sum1stColP

Option Weight Matrices respect parameter k, O[n][n][m]

1. Algorithm
   a. User input raw data, matrix R[m][n], options (n) respect parameters (m)
   b. Calculate the options matrix O respect parameter k
      i. If preference is higher
      ii. If preference is lower

2. Pseudo code
   a. For i in range (m):
      For j in range (n):
      R[i][j] = r[i][j]
   b. For k in range (m):
      For i in range (n):
      For j in range (n):
      If Preference == 1:
      If R[k][i] >= R[k][j]:
      $O[i][j][k] = 1 + \frac{2*(R[k][i]-R[k][j])}{\sigma}$
      If R[k][i] < R[k][j]:
      $O[i][j][k] = \frac{1}{1+\frac{2*(R[k][j]-R[k][i])}{\sigma}}$
      If Preference == 0:
      If R[k][i] <= R[k][j]:
      $O[i][j][k] = 1 + \frac{2*(R[k][i]-R[k][j])}{\sigma}$
      If R[k][i] > R[k][j]:
      $O[i][j][k] = \frac{1}{1+\frac{2*(R[k][j]-R[k][i])}{\sigma}}$

Utility Score, S[n]

1. Algorithm
   a. Normalize elements in O dividing by the summation of column respect each parameter k
   b. Divided each elements in column by the maximum elements in that column get optionWeight[n][m]
   c. Multiply optionWeight by parameterWeight to get utility score S, and store the max score and the index(the option)
   d. Normalize U by diving the sum of the elements.
      i. The first entry will correspond to the first option and will continue in the order of options.

2. Pseudo code
   a.
```
For j in range (m):
        For i in range (n):
                sum1stColO [j] += O[i][0][j]
For j in range (m):
        For i in range (n):
                optionWeights[i][j] = O[i][0][j] / sum1stColO [j]
                if optionWeights[i][j] > max[j]:
                        max[j] = optionWeights[i][j]
```
   b.
```
For j in range (m):
        For i in range (n):
                optionWeights[i][j] /= max[j]
```
   c.
```
For i in range (n):
        For j in range (m):
                S[i] = optionWeights[i][j] * parameterWeights[j]
                if(S[i]>optimalScore)
                        optimalScore = S[i];
                        optimalOption = i;
        sumS += S[i]
```
   d.
```
For i in range (n):
        S[i] /= sumS
```

Confidence Scores, SDR[m][n-1]

1. Algorithm
   a. Normalize elements in O dividing by the summation of the optimal option column (z)respect each parameter k
   b. After changing in the Option Weight Matrices, E, the new utility score of option will pass the new utility score of optimal option. How many stander deviation, SDR[m][n-1], change in the raw data depends on the range of original raw date
2. Pseudo code
   a. For j in range (m):
      For i in range (n-1):
         sumC[j] += O[i][z][j]
   b. For i in range (m):
      For j in range (n):
         If (j != z):
            E = $\frac{sumC^2[i]*(S[z]-S[j])}{(1-O[j][z][i]+sumC[i])*W[i]-sumC[i]*(S[z]-S[j])}$
            If (O[j][z][i] >= 1):
               SDR[i][j] = E
            Else:
               If (O[j][z][i] + E <= 1):
                  SDR[i][j] = $(\frac{1}{\mathrm{O[j][z][i]}} - \frac{1}{\mathrm{O[j][z][i]}+E}) * \frac{1}{2}$
               Else:
                  SDR[i][j] = $[\frac{1}{\mathrm{O[j][z][i]}} + (\mathrm{O[j][z][i]} + E) - 2] * \frac{1}{2}$