P U M .:. C

PUMaC Scoring System

Motivation: For the purposes of aggregating team scores and overall individual scores, individual subject test scores should be calculated in such a way that someone equally good at subject test A and subject test B should get approximately the same score whether they take A or B. Otherwise, too much of a team's/individual's score is left up to luck. The scoring system outlined below, which was inspired by a scoring system implemented in 2016 for my high school's math competition, helps mitigate the luck factor.
– Eric Neyman, 2017 PUMaC director

- **For rankings/scores within subject tests:** Each subject test problem is weighted according to the formula

  $$1 + \ln\left(\frac{\text{number of people taking subject test}}{\text{number of people who got the problem right}}\right)$$

  A competitor's subject test score is the sum of the weights of all the problems that the competitor got right.

- **For overall individual rankings/scores:**

  – Normalizing subject test scores

  ```
  algebra_normalized_scores = ...
          get_normalized_scores_given_exponent(algebra_scores, ...
                       get_normalization_exponent(top_10_algebra_scores, 0.8))
  ```

  This makes it so that the individuals who qualify for individual finals via algebra have an average algebra post-normalization algebra score of 0.8, with the top score being a little above 1. The same is done for combinatorics, geometry, and number theory.

  We can think of the value 0.8 as an indicator of how much weight the subject test scores are given in the overall individual standings. If the 0.8 were replaced by a number very close to 1, this would mean that we decide to differentiate very little between the top 10 subject scores, thus not weighting the subject tests very much. If the 0.8 were replaced by a number very far from 1, that would constitute rewarding doing extremely well on the algebra test; it would essentially ensure that those that do the very best on the subject tests get the top overall individual scores. The 0.8 serves as a middle ground.

  Also, the higher the number in place of 0.8, the more well-rounded it serves for a competitor to be. Doing well on two subject tests is better than doing extremely well on one and mediocrely on another.

  – Normalizing individual finals scores

  ```
  indiv_finals_normalized_scores = get_normalized_scores(indiv_finals_scores, 0.7)
  ```

  – An individual's final score is calculated as the sum of the individual's two normalized subject test scores and 2 times the individual's normalized individual finals score.

- **For team round scores:** Problem weights will be given rather than calculated, because this might be necessary for the strategy/game theory component that is characteristic of PUMaC team rounds.

- **For power round scores:** Problem weights will be given.

- **For aggregate team score:**

  – Individual test score contribution

  ```
  algebra_normalized_scores = get_normalized_scores(algebra_scores, 0.3)
  ```

  This makes an average participant's score worth just under 0.3 times as much for the team as the best-in-subject-test participant's score. The same is done for combinatorics, geometry, and number theory.

```
team_individual_score_components = list, for all teams, of the sum over four
    subject tests of the sum of the 5 highest normalized individual scores
    on the team for that subject test

normalized_team_individual_score_component = ...
    get_normalized_scores(team_individual_score_components, 0.5)
```

  - Team test score contribution

```
normalized_team_round_component = get_normalized_scores(team_round_scores, 0.5)
```

  - Power Round score contribution

```
normalized_power_round_component = get_normalized_scores(power_round_scores, 0.5)
```

  - A team's final score is 2 times individual score component plus 2 times its power round component plus its team round component.

- Function implementations

```
function get_normalized_scores(list_of_scores, desired_average_normalized_score):
    normalized_scores = []
    new_scores = pre_normalize(list_of_scores)
    binary search for the exponent p such that the average of all scores in
        new_scores of score^p is equal to desired_average_normalized_score
    for score in new_scores:
        normalized_scores.append(score^p)
    return normalized_scores
end function

function get_normalized_scores_given_exponent(list_of_scores, exponent):
    new_scores = pre_normalize(list_of_scores)
    normalized_scores = []
    for score in new_scores:
        normalized_scores.append(score^exponent)
    return normalized_scores
end function

function get_normalization_exponent(list_of_scores, desired_average_normalized_score):
    new_scores = pre_normalize(list_of_scores)
    binary search for the exponent p such that the average of all scores in
        new_scores of score^p is equal to desired_average_normalized_score
    return p
end function

function pre_normalize(list_of_scores):
    new_scores = []
    highest = max(list_of_scores)
    for i in range(len(list_of_scores)):
        new_scores.append(ln(list_of_scores[i]/highest + 1.00736)/ln(2.08235))
        # The constant 2.08235 does not matter very much but should definitely be a
        # number that is greater than 2 by a small but not tiny amount. This
        # ensures that the adjusted highest score is slightly less than 1 and
        # that all scores are nonnegative. Similarly, 1.00736 should be greater
        # than 1 by a small(er) but not tiny amount.
    return new_scores
```