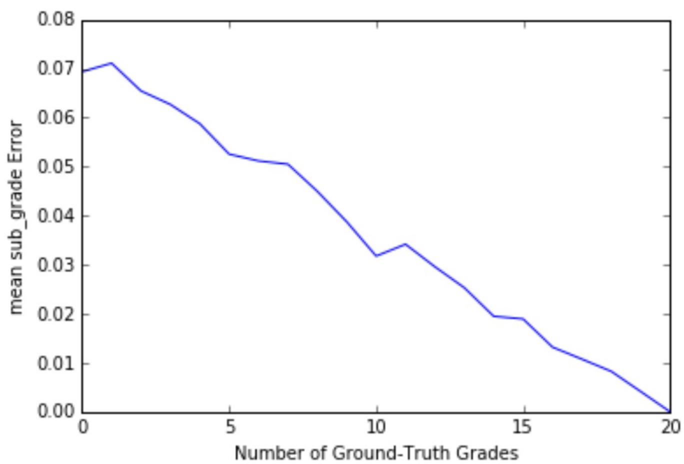
In [2]:
```
%matplotlib inline

from pprint import pprint
from peer_review import *
from vancouver_simulations import *
import numpy as np
import matplotlib.pyplot as plt
import operator
```

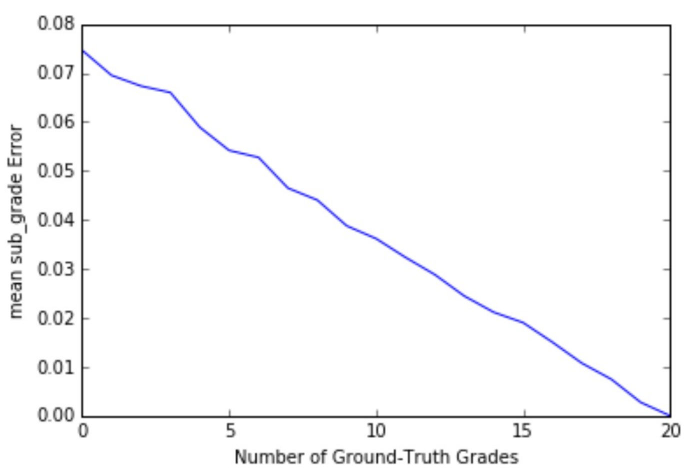
Vancouver Iterations

Upon working more with the code and moving it into my IDE, I discovered that I had neglected to pass the number of iterations to Vancouver appropriately. I have corrected that error, but per the below it doesn't look like it makes much of a difference.

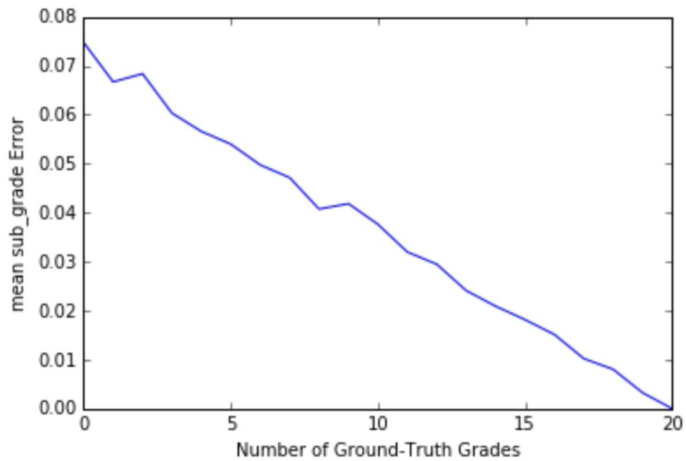
In [6]:
```
plot_stats('mean', 'sub_grade', (random.randint, 1, 5), num_trials=40, use_cover=Tr
ue, vancouver_steps=10)
```



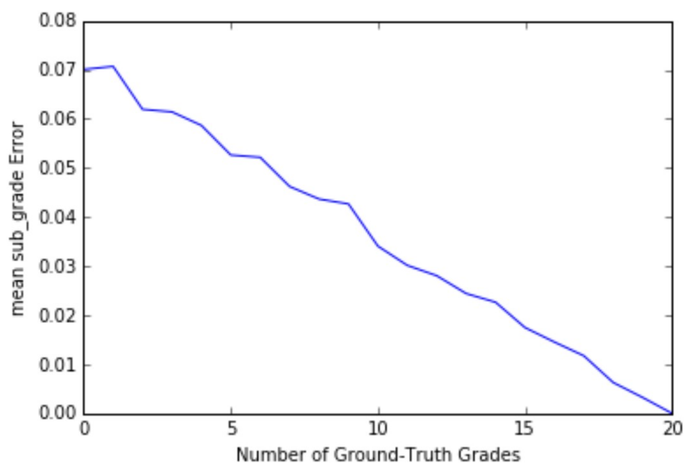
In [7]:
```
plot_stats('mean', 'sub_grade', (random.randint, 1, 5), num_trials=40, use_cover=Tr
ue, vancouver_steps=20)
```



In [8]:
```
plot_stats('mean', 'sub_grade', (random.randint, 1, 5), num_trials=40, use_cover=Tr
ue, vancouver_steps=30)
```



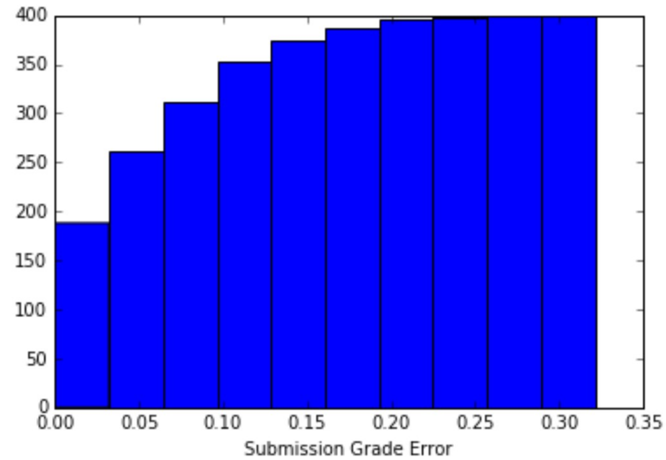
In [9]:
```
plot_stats('mean', 'sub_grade', (random.randint, 1, 5), num_trials=40, use_cover=Tr
ue, vancouver_steps=1)
```



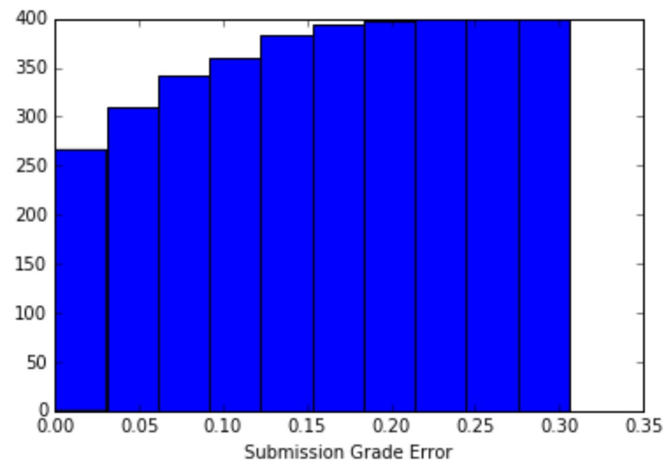
It looks like over a uniform distribution of graders, even a single iteration of Vancouver is pretty much the same as multiple iterations of it. My next step will be to examine other distributions.

Attempt to Elicit Non-Linear Decrease in Errors

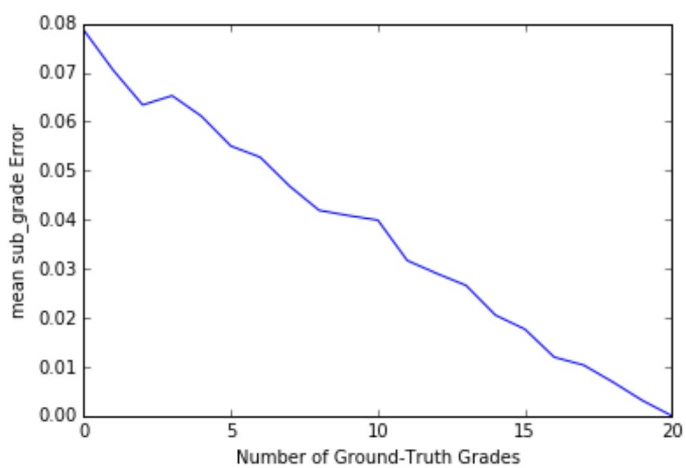
In [10]:
```
plot_histogram(peer_quality=(random.randint, 1, 5), num_truths=5)
```



In [11]:
```
plot_histogram(peer_quality=(random.randint, 1, 5), num_truths=10)
```

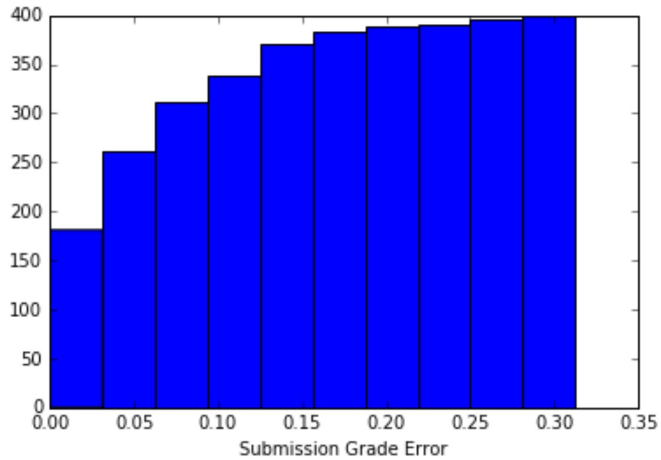


In [12]:
```
plot_stats('mean', 'sub_grade', (random.choice, [1, 5]), num_trials=40, use_cover=T
rue, vancouver_steps=10)
```

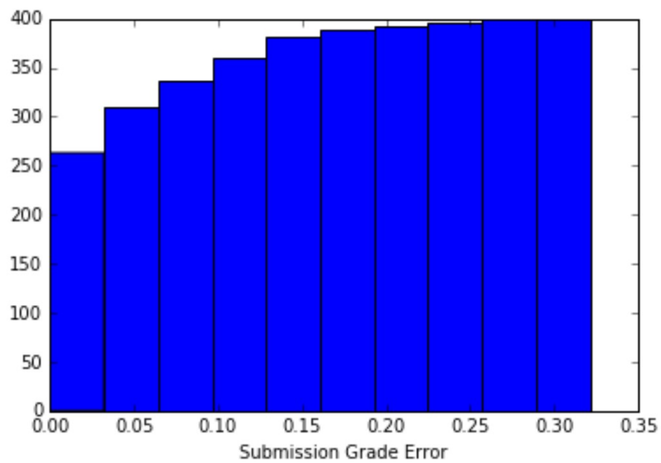


Distribution at Extremes

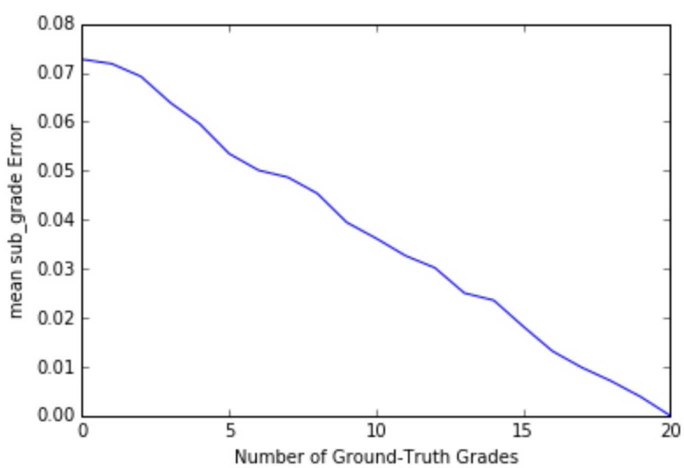
In [13]:
```
plot_histogram(peer_quality=(random.choice, [1, 5]), num_truths=5)
```



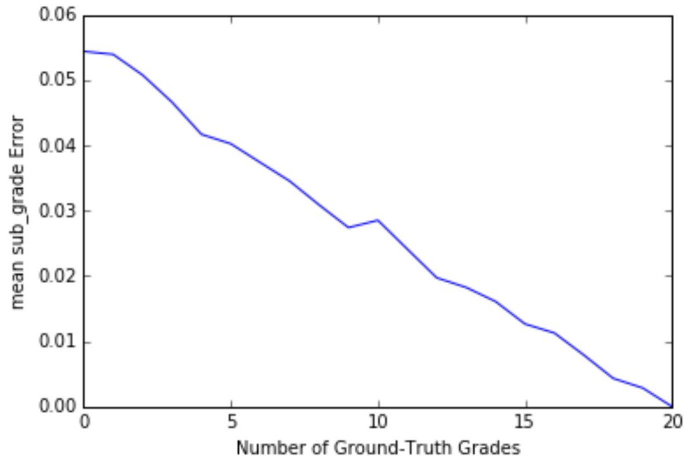
In [14]:
```
plot_histogram(peer_quality=(random.choice, [1, 5]), num_truths=10)
```



In [15]:
```
plot_stats('mean', 'sub_grade', (random.choice, [1, 5]), num_trials=40, use_cover=T
rue, vancouver_steps=10)
```

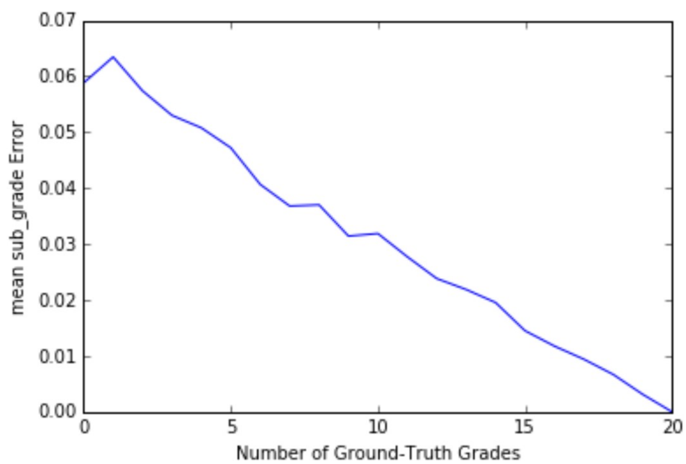


In [16]:
```
plot_stats('mean', 'sub_grade', (random.choice, [1, 10]), num_trials=40, use_cover=
True, vancouver_steps=10)
```

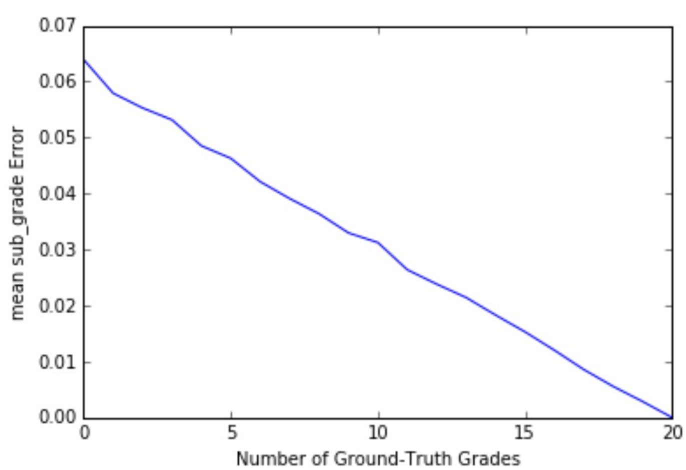


Skewed Distribution

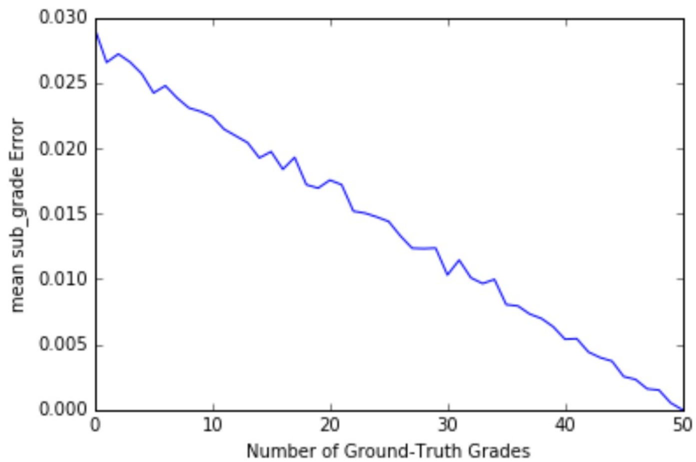
In [17]:
```
plot_stats('mean', 'sub_grade', (random.choice, [1, 5, 5, 5]), num_trials=40, use_c
over=True, vancouver_steps=10)
```



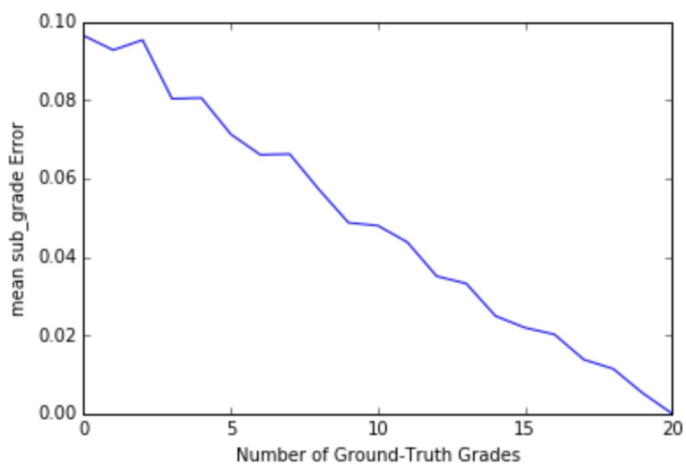
In [18]:
```
plot_stats('mean', 'sub_grade', (random.choice, [1, 5, 5, 5]), num_trials=100, use_
cover=True, vancouver_steps=10)
```



In [3]:
```
plot_stats('mean', 'sub_grade', (random.choice, [1, 10, 10, 10]), num_trials=20, us
e_cover=True, vancouver_steps=10,
          num_subs=50, num_grades_per_sub=5)
```



In [20]:
```
plot_stats('mean', 'sub_grade', (random.choice, [1, 1, 1, 5]), num_trials=20, use_c
over=True, vancouver_steps=10)
```

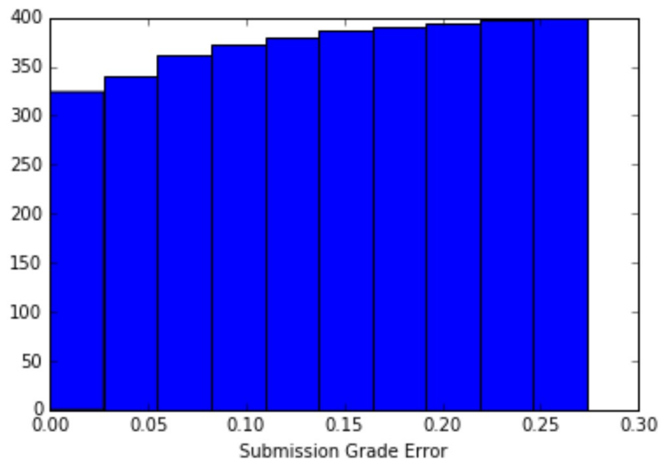


The results are linear for all three of these distributions. The fluctuations in quality with no ground truth are probably due to the differences in grader quality, with distributions that have more high-quality graders ending up with better grades.

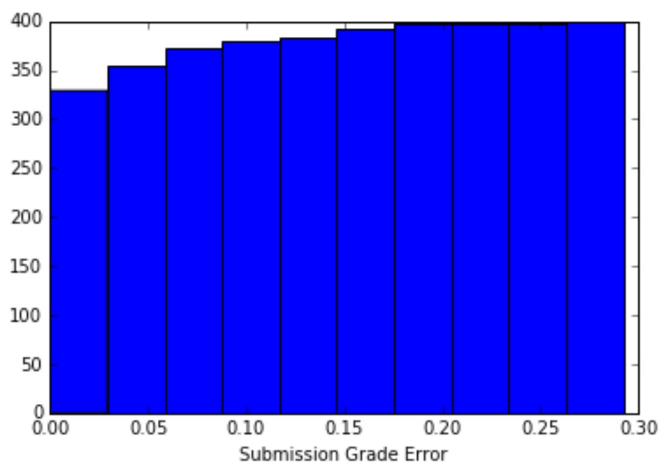
Attempt to Find Better Algorithms for Order of Ground-Truth Grades

I will start this examination by finding out if grading in the order of most error or most variance, or a multiple of the two, has any effect.

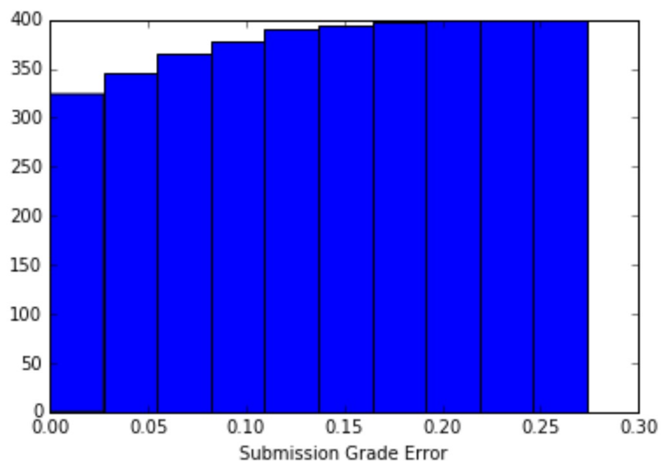
In [21]:
```
def alg(t, init, actual):
    scores = init[0]
    qualities = init[1]
    omni_scores = actual[0]
    true_qualities = actual[1]

    sub_score_error = [abs(scores[submission][0] - 0.5) for submission in scores]
    sub_var_error = [abs(scores[submission][1] - omni_scores[submission][1]) for su
bmission in scores]
    grader_var_error = [abs(qualities[grader] - true_qualities[grader]) for grader
in qualities]

    return max(sub_grade_error.iteritems(), key=operator.itemgetter(1))[0]

plot_histogram(peer_quality=(random.choice, [1, 5]), num_truths=15, grading_algorit
hm=alg)
```



In [22]:
```
plot_histogram(peer_quality=(random.choice, [1, 5]), num_truths=15)
```



In [23]:
```
def alg2(t, init, actual):
    scores = init[0]
    qualities = init[1]
    omni_scores = actual[0]
    true_qualities = actual[1]

    sub_score_error = [abs(scores[submission][0] - 0.5) for submission in scores]
    sub_var_error = [abs(scores[submission][1] - omni_scores[submission][1]) for su
bmission in scores]
    grader_var_error = [abs(qualities[grader] - true_qualities[grader]) for grader
in qualities]

    sub_var = [scores[submission][1] for submission in scores]

    return max(sub_var.iteritems(), key=operator.itemgetter(1))[0]

plot_histogram(peer_quality=(random.choice, [1, 5]), num_truths=15, grading_algorit
hm=alg2)
```



Choosing by highest grade error and choosing by highest variance appear to have minimal effects compared to choosing randomly after the cover. I have run these trials at a couple different num_truths values, and it looks like all three methods are pretty even.

In [ ]: