Coins in a Pot

ISYE-6739 - Statistical Methods

Kadhir Umasankar

December 7th, 2021

Abstract

TODO

1 Introduction

Analysis of games can be done by running many simulations. Code must be written to convert the problem at hand into a simulation that the computer can run, and statistics of interest can be recorded over many runs. By running thousands of such simulations, the estimations of statistics will begin to converge to the parameters' actual values.

In this report, analysis of the "Coins in a Pot" game will be described. In this game, the two players and pot start with a certain number of coins. The players take turns rolling a die, and they perform one of the following actions based on the number they rolled:

Number on Die	**Action**
1	Player does nothing
2	Player takes all coins in the pot
3	Player takes half of the coins in the pot (rounded down)
4, 5, 6	Player puts a coin in the pot

The game ends when a player has no coins left and must put a coin in the point. The number of cycles that the game lasts must be recorded, and its expected value and distribution are to be found.

Following these rules, some Python code was written. The number of cycles that were played until a player ran out of coins was recorded in a list, and this process was repeated many times. The expected value of the number of cycles will be found, and the distribution of the number of cycles will be plotted.

TODO: What's coming in the rest of the report (along with "this will be talked about in this section")

2 Data Analysis

3 Results and Discussion

Table 1 was created by using Equation **??** to find the critical stress intensity factors, K_C, for different values of initial crack lengths, a_0, and different specimen thicknesses. Using the same data points, a plot of the fracture toughness for different initial crack lengths was created, which can be seen in Figure **??**. A plot of the average critical stress intensity factor against the specimen thicknesses can be seen in Figure **??**, and the negative trend seen in this figure is intuitive, as average toughness decreases with increasing specimen thickness.

Thickness	a_0 **(in)**	K_C
0.5 in	0.89	37965.8244
	0.986	37908.5895
	0.905	30781.3273
	1.061	41217.1462
	1.006	38510.6926
	0.851	34975.1817
0.25 in	0.719	37955.1178
	0.9345	51430.7034
	0.9585	37061.6005
	0.82	48584.4641
	0.703	35624.8686
	0.772	54836.7813
0.125 in	0.987	57274.8526
	1.105	51956.9193
	1.118	51847.9702
	0.918	36215.4784
	0.819	46849.6807
	0.9385	30393.6034

Table 1: Table of critical stress intensity factors (K_C) for different initial crack lengths (a_0)

Table 2 shows the evaluation of Equation **??** for all of the samples. From this table, we can see that since some values of a satisfy the condition for K_{IC}, the thicknesses of the specimens are too small to satisfy the condition. Therefore, to find K_{IC}, we would have to test thicker specimens.

Thickness	a_0 **(in)**	$2.5(\frac{K_C}{\sigma_{ys}})^2$
0.5 in	0.89	1.1091
	0.986	1.1057
	0.905	0.7290
	1.061	1.3072
	1.006	1.1411
	0.851	0.9412
0.25 in	0.719	1.1084
	0.9345	2.0353
	0.9585	1.0569
	0.82	1.8162
	0.703	0.9765
	0.772	2.3138
0.125 in	0.987	2.5241
	1.105	2.0771
	1.118	2.0684
	0.918	1.0092
	0.819	1.6888
	0.9385	0.7108

Table 2: Table showing $2.5(\frac{K_C}{\sigma_{ys}})^2$ for all of the samples

When examining the fracture surfaces, the fracture line is perpendicular to the loading direction, which is the plane-strain condition. Further along the specimen, the fracture line is at a 45° angle to the loading direction, which is the plane-stress fracture mode. This is intuitive, because 45° is the angle of maximum shear strain.

Plots showing the load v.s. displacement for the different specimens can be seen in Figures **??** to **??**. It can be noted visually that the slopes of the linear parts of the curves are approximately equal.

Figure **??** shows a plot of the compliance values obtained for each specimen thickness as a function of the initial crack length. This proves the observation made earlier: that the slopes of the linear parts of the curves in Figures **??** to **??** are approximately equal.

4 Conclusions

In this lab, fracture tests were conducted on samples of 2024-T3 Aluminum. Data acquired during the fracture tests was processed to find the critical stress intensity factors, load v.s. displacement plots, and compliance for each sample. By computing these properties, it was found that increasing the thickness of the material reduced the average critical stress intensity, and also brought the K_C value closer to K_IC. It was also found that the compliance values for the material were roughly within the same range.

Through this lab, the importance of fracture testing materials was demonstrated. Knowledge of fracture toughness is very important in industry as it will provide a lot of information about the suitability of a material for a specific use.

Appendix A

Python Code

```python
import math
import random
import statistics
import subprocess
import time

import matplotlib.pyplot as plt
from bashplotlib.histogram import plot_hist


class Player:
    def __init__(self, id, coins):
        self.id = id
        self.coins = coins


def print_scores():
    print("-------------")
    print("| A |Pot| B |")
    print("-------------")
    print(f"| {player_A.coins} | {pot.coins} | {player_B.coins} |")
    print("-------------")


def print_ascii_hist():
    if len(cycle_list) > 0:
        plot_hist(
            cycle_list, pch=".", bincount=math.ceil((max(cycle_list) - min(cycle_list)))
        )
        # print(f"μ = {mu_list[-1]}")
```

```
if __name__ == "__main__":
    cycle_list = []
    mu_list = []
    num_trials = 100000
    for i in range(num_trials):
        player_A = Player("A", 4)
        player_B = Player("B", 4)
        pot = Player("pot", 2)
        counter = 0
        while True:
            if counter % 2 == 0:
                current_player = player_A
            else:
                current_player = player_B
            num = random.randint(1, 6)
            if num == 1:
                # print(f"{current_player.id} rolled {num}, and does nothing")
                pass
            elif num == 2:
                current_player.coins += pot.coins
                pot.coins = 0
                # print(f"{current_player.id} rolled {num} and took all the coins from
            elif num == 3:
                current_player.coins += pot.coins // 2
                pot.coins -= pot.coins // 2
                # print(f"{current_player.id} rolled {num} and took half of the coins
            else:
                if current_player.coins == 0:
                    # print(f"{current_player.id} rolled {num} but has 0 coins. The ga
                    cycle_list.append(counter // 2 + 1)
                    break
                current_player.coins -= 1
                pot.coins += 1
                # print(f"{current_player.id} rolled {num} and put one coin in the pot
            counter = counter + 1
        # mu_list.append(statistics.mean(cycle_list))
        if (i + 1) % 2000 == 0:
            subprocess.run(["clear", "-x"])
            print_ascii_hist()
            print(
                f"{i+1} of {num_trials} runs complete ({math.ceil(i/num_trials * 100)}%)
            )

    plt.hist(cycle_list, bins=math.ceil((max(cycle_list) - min(cycle_list))))
    plt.xlabel("Cycles")
    plt.ylabel("Frequency")
    plt.title(f"µ = {statistics.mean(cycle_list)}")
    plt.show()
```

TODO it overflows