Initial Project Questions/Comments:

Histogram Colors:
- 2 corrective vectors (x and y directions) per data point
- sort them largest to smallest by magnitude
- plot the magnitudes on the y-axis, sorted indices on the x-axis
- color positive values green, negative values red
  - used to display that the data’s sign distribution is random with zero bias
  - shows that 1 bit per compressed value should be designated and fixed for sign
    - because it will be positive 50% of the time, negative the other 50%
    - no gain by variable length encoding

Loss of Accuracy through Linear Estimation:
- Consider:
  - quantized values from 0 to 4095
  - A sequence of data points \{0, 4095, 0\}
  - A linear estimate of the 3\textsuperscript{rd} point is 8190
  - The corrective value is -8190
  - requires 1 more than the original number of quantized bits (plus sign)
    - Many times more than the expected compressed amount of bits
    - Greater than not compressing with linear estimation and correction
    - Or, Maybe not.
      - Allow both + and – values to wrap around integer range
      - $\rightarrow$ New worst case = Difference of half of quantized range

Interpolating Predictor

Motivation: Send a crude sampling of the data points first (perhaps every other point), then estimate the intermediate points through interpolation and only send the corrections.
**Develop Cubic Interpolating Predictor:**

**Strategy:** fit cubic to 4 points
- Polynomial $P(t) = \text{constraint}$
- Interpolate Constraints:
  
  - $P(t_A) = A$
  - $P(t_B) = B$
  - $P(t_C) = C$
  - $P(t_D) = D$

**How to choose t:** Use arc length parameterization
- If not uniform: Use separation distance
  - However, this is difficult to solve
- Assume uniform separation
  - Can use any separation distance
  - Use $t_A = -3$, $t_B = -1$, $t_C = 1$, $t_D = 3$
  - Allows the center point, the interpolated point, to be evaluated as $P(0)$

**Solve:**

$$f(t) = at^3 + bt^2 + ct + d$$

Find: $d = f(0)$

$$f(-1) = B = -a + b - c + d \quad f(-3) = A = -27a + 9b - 3c + d$$

$$f(1) = C = a + b + c + d \quad f(3) = D = 27a + 9b + 3c + d$$

$$\frac{B + C}{2} = b + d \quad \Rightarrow \quad \frac{A + D}{2} = 9b + d \quad \Rightarrow \quad 9\left(\frac{B + C}{2}\right) - \left(\frac{A + D}{2}\right) = 8d$$

$$d = \frac{9}{8}\left(\frac{B + C}{2}\right) - \frac{1}{8}\left(\frac{A + D}{2}\right) \quad \Rightarrow \quad d = \frac{B + C}{2} + \frac{1}{8}\left(\frac{B + C}{2} - \frac{A + D}{2}\right)$$

Exercise: Use a cubic to extrapolate a curve, just as before using $0^{th}$, $1^{st}$, $2^{nd}$, etc, order predictors, and compare the two.
Encoding Data

We have symbols from Set \{s_0, s_1, s_2, \ldots\}, how to encode them?

To send a LONG string
- Assume same probability of symbol occurrence
- Assume \(2^n\) symbols
- Encode the symbols by their bit reference
- Cost per symbol? \(\text{cost} = \text{bits per symbol} = n \text{ bps}\)

Cost:
- Need to know \(n\)
- Dictionary (the hidden cost)
  - \(S_i\) represent integers
  - Only useful if sending the same integer many times
  - If only send once, why encode the data?; encoded bits are extra bits to send

Don’t Assume Same Probability:
- Merge Symbols, allowing a smaller bit length to reach more frequent symbols.
- Variable Length Encoding
**Variable Length Encoding**

*Theoretical idea:*

- $h = \text{height} = \# \text{bits}$
- $\text{reference}\# \approx \frac{1}{\text{probability(symbol)}}$

A symbol with larger probability should be given a smaller reference number.

- $\log_2(\text{reference}\#) = \# \text{bits} = h$
- $h = \log_2\left(\frac{1}{p}\right)$

A symbol with larger probability should have a smaller cost.

**Expected Cost**

- Expected Actual Cost = $\sum h_i p_i$ h based on actual, used, tree structure.
- Expected Theoretic Cost = $\text{Entropy} = \sum p_i \log_2\left(\frac{1}{p_i}\right)$
  - Theoretical Only
    - Could be more (dictionary cost, …)
    - Could be less (specific off-probability strings, …)
Example:

\[ P_0 = 0.7, P_1 = 0.1, P_2 = 0.1, P_3 = 0.1 \]

Type 1:

\[
\begin{align*}
\log(4) & = 2 \text{ bps} \\
1 \cdot 0.7 + 2 \cdot 0.1 + 3 \cdot 0.1 + 3 \cdot 0.1 & = 1.5 \text{ bps}
\end{align*}
\]

\[ \rightarrow \text{Structure/Probability affects the expected cost} \]

**Huffman Tree Algorithm:** One method for sorting the tree based on symbol probability

1) Keep all \( p_i \) sorted from smallest and up at each step

\[ P_1 = 0.1 \quad P_2 = 0.1 \quad P_3 = 0.1 \quad P_4 = 0.7 \]

2) Merge the left most two and resort, repeat until all are merged

A)

\[
\begin{align*}
0.1 & \quad 0.2
\end{align*}
\]

B)

\[
\begin{align*}
0.3 & \quad 0.7
\end{align*}
\]

C)

\[ \text{Note: Only good if symbols are sent 3, 4, 5, \ldots \text{ times.}} \]

\[ \text{Probably not good for a set of floating point corrections, where there is minimal repetition.} \]
How to Encode/Decode the binary tree?

Send path strings to be reconstructed directly

- Reconstruct directly based on the data, which is the directly represented tree
- How to discriminate the strings? → One method is to use padding
- Cost \( \geq nh = n \log(n) + n(Padding\text{Cost}) \), for the best case, even, tree
- **High Cost** to directly send the tree structure

Encode/Decode using recursion

One method: *(Note: This may be slightly different than the actual method seen in class)*
- Start at the left bottom, then work up, taking the bottom left nodes first.
- A “0” is a final node (non-branching)
- A “1” binds the previous two nodes

- **Cost** = \( 2n - 1 \), Only 1 bit per node, 2n-1 nodes per tree