I hope it is not too precious to include an explanation of the algorithm as example text for the algorithm. The main function that does the heavy lifting is calculate-breakpoints(). This function runs in O(n-squared) time, iterating once over the words in our sample text, and then comparing each word to a subset of potential line breaks. A more detailed explanation of this algorithm follows. We begin by setting the first breakpoint at the beginning of the text. We then iterate through the words of the text, at each step assessing the line starting at the first breakpoint, and extending up through the word we are on. Each of these lines is assigned a score, determined by the stretching or shrinking factor necessary to fit this line to the prescribed length (as set in 'target'). If the spaces must be stretched or shrunk beyond their capacity (or tolerance), the line is disregarded as unfeasible. However, if a line is feasible, its score is saved, and a new line in considered, now extending from the second breakpoint (should it exist) to our current word. Once all possible breakpoints are assessed, the one with the lowest cost is appended to the list 'candidate-breaks'. We continue in this fashion, updating 'candidate-breaks' with growing sequences of possible breakpoints until we reach the end of the text. When the algorithm reaches the final word, it again calculates the optimal breakpoints for that word. It then considers all remaining sequences in 'candidate-breaks'. Each of these is assigned a cumulative score corresponding to the total stretching/shrinking factor of all of its constituent lines. The sequence with the lowest score is chosen, and its corresponding breakpoints are applied to the text.