**Creation of a numerical solution:**

To apply our mathematical model to real-life example, we must first create a numerical tool that will generate a large number of outputs. After we have created this tool, we must create another tool that will compare the current ongoing ski season to the large number of tools we have generated, and then give possible predictions about the future snow depth. To build these two tools, we will be coding programs using the Python programming language. This language is highly polyvalent thanks to the many libraries made for it which can be used to expand our programming horizons. Furthermore, it is free and fully open source, this might be useful for commercial applications. In comparison, other mathematics-oriented programming languages such as MATLAB cost thousands of dollars in base software and extensions every single year.

Initial Settings:

We must first set some initial parameters that will help construct the rest of the code.

This starts by importing the Pandas library which we will use throughout the program. We must also set the number of models we want to generate thanks to the *n* variable. Furthermore, we also define the number of intervals we would like to study, we will usually choose 100, as a ski season usually includes about 100 days. We also define two essential data frames, *dg* - which houses all the generated models - and *df* – which is our template sheet and includes our columns -.

Text

Description automatically generated

Function Calculation:

The first part that our program must feature is the ability to calculate multiple images of our function and store it.

We will be iterating for *n* times the next 3 steps (Function Calculation, Brownian Motion, Depth Model), creating *n* times seriess. We must also set parameters which are unique to the generation of models. To start off, *total* indicates the function interval, most often 1 when evaluating a whole season, might be 0.5 when evaluating only the first part of it. *dt* is the time interval we will use in the function with *t* being the current time. We then iterate *T* times the process of creating values for each *t* of the Beta function. After finding the image of the function, we add it to the *df* data frame in the ‘Beta(t)’ column.

Graphical user interface

Description automatically generated with medium confidence

Brownian Motion:

The second aspect of our program consists in creating the stochastic part of our model. This will be done by finding a numerical solution to a Wiener process. We start by importing a function from the *scipy.stats* library called *norm*, standing for normal distribution which is used in the choice of random variable. We must also define two initial parameters, *x* indicates the initial state of the Brownian motion and *dev* indicated the deviation the model will feature, as mentioned, this is a mere direction, as because of stochasticity, the function could largely exceed this indicator. We then iterate for the number of intervals previously mentioned the creation of Brownian motion function images. The *norm.rvs* allows us to generate random continuous variables following our as a *scale* or the standard deviation of the function. We then add this generated variable to our data frame under the ‘W(t)’ column.

Text

Description automatically generated with medium confidence

Depth Model:

The third part of the program is to add these two parts of our function to get the final result of our function d(t). We first have to adjust some values, as at the initial stages of the Beta function, the values are usually close to 0 and the Brownian process might issue negative values therefore outputting negative depth values. To avoid this, we first verify if the value of Beta(t) + W(t) is larger than 0, if not we assign D(t) the value of 0 and add it to the data frame in the ‘D(t)’ column. In the opposite case, we simply add the two values and copy them to the ‘D(t)’ column. The last part is to add this season to our data frame of all generated seasons. To achieve this, we iterate in *T* and copy all the values into our *dg* data frame in the *k* column and the *i* row.

Text

Description automatically generatedShape

Description automatically generated with low confidence

To illustrate the stochasticity of our depth model, we generate 50 unique models and plot them with a *dev* of 0.3 and 0.5 using 100 intervals. To plot these graphs, we will be exporting our *dg* data frame into an Excel sheet using the *datatoexcel* function and plotting them there.

Chart

Description automatically generatedChart

Description automatically generated

Another way of handling our deviation, it to add a coefficient that multiplies our Wiener process, that way we can get patterns that can be concentrated thanks to our *dev* but then, because of our amplification of the phenomena, the rare cases will have their values amplified, making it more interesting for our model. To apply this, we will modify our code to add a multiplier to our Wiener process.

Text

Description automatically generated with low confidence

To illustrate this change, here is a graph of 50 different generated models using these modified parameters:

Chart

Description automatically generated

This change of parameters might be useful when trying to model different situations and mountains results, also one might want to model more extreme models, whereas some others might want to model a function closer to the yearly average.

Correlation to other seasons:

The final and most complicated part of our model is to find similarities between the ongoing season and our previously generated stochastic models. To achieve this, we will be comparing all the outputs to the current ongoing season, note that we will only be considering our outputs up to the equivalent of today’s date on the ongoing season. As our stochastic model does not have a specific scale and must be adapted to each snow resort, we must use a comparison method that takes into account the similarities in variations and not in absolute value. Therefore, we will be calculating the similarities thanks to a Pearson’s Correlation Ratio and at the end of the process, we will be seeing which model has the closest resemblance to the ongoing ski season which will allow us to make predictions of future snow depth.

The first step is to extract the current season data, this can be done either by importing an Excel sheet containing the data, or by importing a CSV file. In either case, both are converted to a Pandas data frame – *ds­* -, which will allow us to handle them efficiently. We might have to modify the data plot of the current season so that it is compatible with our model, such as removing the beginning of end of the plot as it might include other seasons than winter. We also import an Excel sheet – *corrressult* – which is a template for our results.

The next step is to iterate in *n* the correlation calculation between this year’s data and the depth data we have generated. After calculating it using a Pearson’s Correlation, we copy the result into our answer sheet, under the model number.

After doing this, order the rows in function of the correlation ratio, from highest to lowest. We then store the 5 first models and copy them into a separate data frame. We then print a graph using ---- that includes the current season and the 5 models with a higher correlation. We then see what the 5 paths evolve in the future giving us an idea of how snow depth might evolve in the coming days.