THESIS

MODELING OF CHANNEL STACKING PATTERNS CONTROLLED BY NEAR WELLBORE MODELING

Submitted by

Luis Carlos Escobar Arenas (he/him)

Department of Geosciences

In partial fulfillment of the requirements

for the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Summer 2023

Master's Committee:

Advisor: Lisa Stright

Michael Ronayne
Elizabeth Barnes

ABSTRACT

MODELING OF CHANNEL STACKING PATTERNS CONTROLLED BY NEAR WELLBORE MODELING

Reservoir models of deep-water channels rely upon low-resolution but spatially extensive seismic data, high vertical resolution but spatially sparse well log data and geomodeling methods. The results cannot predict architecture below seismic resolution or between well logs. Usually, the data and interpretations that provide constraints for modeling workflows do not capture sub-seismic scale architecture. Therefore, standard modeling methods do not generate models that include details that can impact hydrocarbon flow and recovery. Constraining models to well and seismic data is problematic. Employing measured sections in the Tres Pasos Fm. (Magallanes Basin, Chile) is feasible to predict deep-water channel architecture, specifically channel stacking patterns with 1D information analogous to well data. This research performed near-wellbore modeling to generate multiple scenarios of channel stacking patterns constrained by machine learning-derived probabilities using (i) conditional Monte Carlo simulation with soft probabilities per channel element within the measured section choosing the highest probabilities for each element (ii) conditional Monte Carlo simulation of channel stacking, (iii) template-based modeling, (iv) forward modeling with Markov transition probabilities without matching to thickness and (v) conditional Monte Carlo simulation constrained to measured section thickness.

Machine learning workflows generate channel position probabilities (i.e., axis, off-axis, margin) within a measured section given the interpreted top/bases of channel elements. These probabilities constitute the input for Monte Carlo simulations capturing channel element stacking patterns at the measured section locations. The most likely 2D channel stacking pattern scenarios defined channel centerline points, and volumes of the individual channel elements can be generated connecting them.

Surface-based modeling offers a way to depict reservoirs of hydrocarbons, water or low-enthalpy geothermal systems in which small-scale heterogeneity needs to be captured explicitly by bounding surfaces because it impacts fluid flow, improving our forecasts of resource exploitation. Furthermore, predicting heterogeneity controlled by depositional architecture is critical for transport and storage capacity in CO_2 reservoirs. The dataset provided and the advent of these flexible and accurate methods to depict the subsurface offer the opportunity to overcome the historical limitations of grid-based models and allow us to assess multi-scale architecture that controls fluid flow. This research aims to show the results of modeling deep-water channels, including a 1D identification of architectural positions and a 2D arrangement of channel stacking patterns.

ACKNOWLEDGEMENTS

This work is a culmination of geologic research conducted by the Chile Slope Systems (CSS) Joint Industry Project, which is a collaboration between the University of Calgary, Colorado State University, Virginia Tech, and industry partners: Chevron, Repsol, Hess, Nexen/CNOOC, ConocoPhillips, BHP Billiton, Anadarko, Equinor, Petrobras, and Shell. Fieldwork, data collection, and interpretation were performed by Brian Romans, Steve Hubbard, Ryan Macauley, Sean Fletcher, and Sarah Southern.

The author thanks the Rocky Mountain Association of Geologists (RMAG), the Society for Sedimentary Geology (SEPM), the American Association of Petroleum Geologists (AAPG) and the Geological Society of America (GSA) for the funds provided.

TABLE OF CONTENTS

CHAPTER 1. RESEARCH MOTIVATION

1.1. Introduction

Deep-water channel systems host significant natural resources. Reservoir models help to predict the range of possible volumes in place. These models are constrained by seismic, that at reservoir depths, has limited ability to resolve the stratigraphic architectural features that control reservoir distribution. Geoscientists can leverage seismic-scale outcrops, thereby bridging the gap between the aerial extent of seismic and bed-scale characterization of stratigraphy.

The confined, amalgamated, seismic-scale channels exposed in the Upper Tres Pasos Fm. (Magallanes Basin, Chile; Fig. 1A) exhibit high net-to-gross (NTG) axis, transitional NTG off-axis to low NTG margin channel fills (Macauley and Hubbard, 2013) and are partitioned by thin, laterally persistent muddy channel base drapes with low permeability (Hubbard et al., 2018) (Fig. 1B). Previous work includes machine learning techniques to predict the stratigraphic architecture at the measured section from vertically high-resolution grain-size curves (Vento, 2020), the generation of a 3D architectural model (Ruetten, 2021) and a 3D seismic-derived probability cube for each architectural channel element from it (Langenkamp, 2021).

The purpose of this thesis is to show the results of multidimensional modeling of deep-water channels, which includes a 1D identification of channel element architectural positions and 2D arrangement of channel stacking patterns using machine learning-derived probabilities, conditional Monte Carlo simulation with soft probabilities and Markov transition probabilities.

1.2. Hypotheses

Reservoir models of deep-water channels rely upon geomodeling methods to combine low vertical resolution but spatially extensive seismic data and high vertical resolution but spatially sparse well log data.

Resulting models predict a range of possible architecture that is likely to exist below seismic resolution or between well logs (Ringrose and Bentley, 2021) as long as architecture is taken into consideration during the modeling process. Nevertheless, the interpretations and attributes that provide constraints for modeling workflows cannot consider sub-seismic scale architecture, and standard modeling methods are limited in their ability to generate realistic architecture and constraining to well and seismic data.

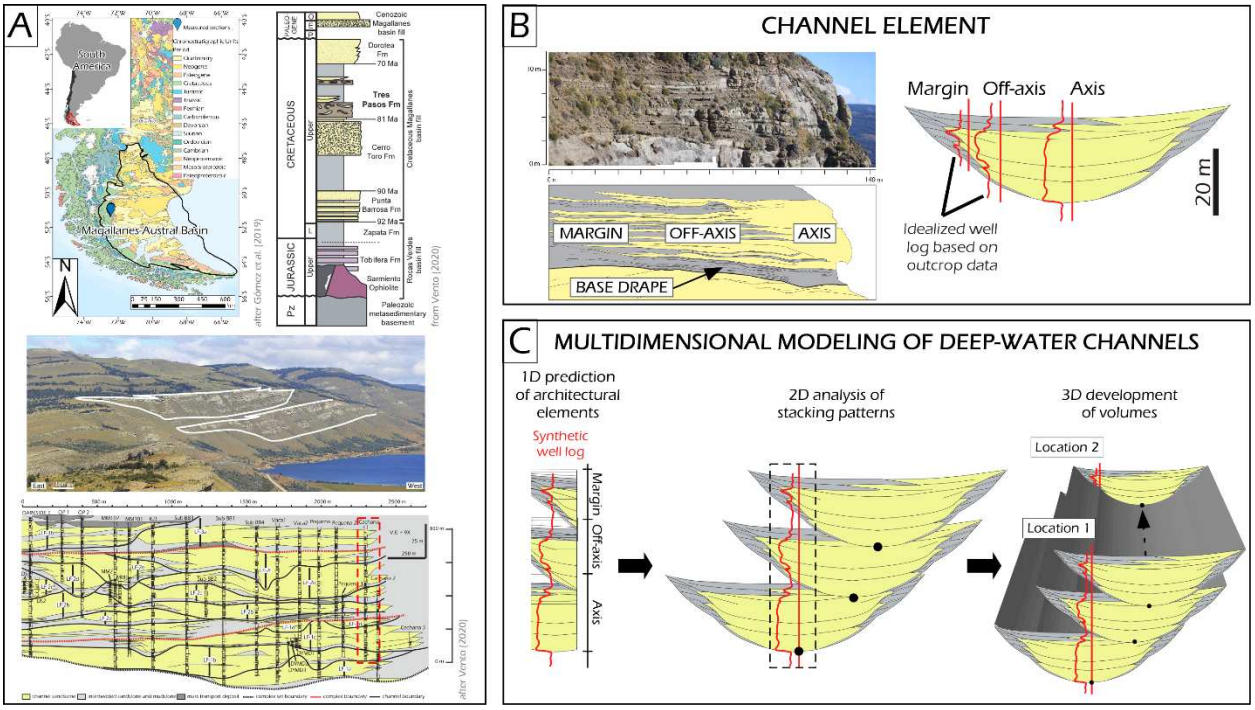


Fig. 1. (A) Geology of the Magallanes (Chile) or Austral (Argentina) Basin (Gómez et al., 2019). The Tres Pasos Fm. occurs as part of the western Cretaceous thrust belt adjacent to the Neogene foreland basin. A generalized stratigraphic column, picture of the outcrops, and stratigraphic correlation of the measured sections are also shown, (B) longitudinal and cross-section view of a channel element, and the theoretical well log response of channel architectural elements (Hubbard et al., 2018), (C) sketch of the proposed workflow to predict channel stacking patterns from spare well data attaining depositional concepts.

We hypothesize that we can leverage >5000 m of measured sections in the Upper Tres Pasos Fm. (Magallanes Basin, Chile) to demonstrate a process of predicting stratigraphic architecture with 1D information analogous to well data (Fig. 1A-C). In this research, we perform near-wellbore modeling to generate multiple realizations of channel stacking patterns constrained by machine learning-derived

probabilities. These results will be the anchor points to correlate deep-water channels between wellbores (Fig. 1C, 3D development of volumes).

1.3. Scientific significance

Currently, the advent of new methodologies to build predictive reservoir models of the subsurface like rule-, surface- or process-based modeling allows us to obtain insights about the evolution of channelized systems, the stratigraphy of small-scale heterogeneities and their impact on fluid flow (Jacquemyn et al., 2019; Pyrcz et al., 2015). Several surface-based models exist for fluvial channels (Pyrcz et al., 2009), shallow marine structures (Jackson et al., 2014; Nordahl et al., 2005), submarine lobes (Pyrcz et al., 2005; Zhang et al., 2009) and submarine channels (Covault et al., 2016; McHargue et al., 2011b; Pyrcz et al., 2015; Stright et al., 2013; Sylvester et al., 2011).

Nevertheless, there are still some remaining challenges in applying surface-based models. Recent endeavors address the conditioning and fitting of the models to well data by performing two-step simulations (Pyrcz et al., 2015) and using artificial neural networks (Titus et al., 2021). Now, we are constraining 2D channel stacking patterns that honor the available data, generating a realistic stratigraphic architectural representation that diminishes the oversimplification imposed by object-oriented models, which cannot match depositional principles and data at the same time (Pyrcz et al., 2015; Pyrcz and Deutsch, 2014).

CHAPTER 2. GEOLOGIC BACKGROUND

2.1. Geological framework

The Southern Patagonian Andes resulted from the complex interaction between the South American, Scotia and Antarctic plates, starting from the Jurassic with the rupture of Gondwana to the ongoing Andean orogeny. The associated tectonic and sedimentary processes created a multi-episodic, retro-arc foreland basin known as the Austral (Argentina) or Magallanes (Chile) Basin (Fig. 2) (Macauley and Hubbard, 2013; Mpodozis et al., 2011; Romans et al., 2011, 2009).

The evolution of the Austral-Magallanes Basin includes two stages. The first stage, related to the rupture of Gondwana, created extension from the middle Jurassic to the Early Cretaceous. Silicic rift-related volcanism is recorded by the Tobífera Fm. (150 - 138 Ma), consisting of pyroclastic rocks intercalated with lacustrine and fluvial-alluvial sediments. This formation occurs as continuous outcrops overlaying the basement or limited within hemigrabens as sedimentary wedges with homoclinal attitudes (Mpodozis et al., 2011). Extension finished with the development of the back-arc Rocas Verdes Basin (Romans et al., 2011), and the bioturbed black shales with thin sandstone beds from the Zapata Fm. (101 Ma) were deposited. They occur as a well-bedded sequence with a lack of sedimentary structures, suggesting deposition in an anoxic, very deep and restricted environment (Mpodozis et al., 2011).

The second stage, related to the tectonic load from the Andean Orogeny has created compression and uplift since approximately 85 Ma. As a result, a retro-arc foreland basin formed (Mpodozis et al., 2011). The Punta Barrosa, Cerro Torro, Tres Pasos and Dorotea formations constitute the Mesozoic lithological record of this stage (Fig. 3).

The Punta Barrosa Fm. (92 – 85 Ma) consists of interbedded sandy turbidites, slurry-flow deposits, and siltstone and occurs as tabular to slightly lenticular packages related to lobe deposition in an unconfined to weakly ponded setting (Mpodozis et al., 2011). The thickness of this formation is about 1000 m and

records the onset of deep-water, turbiditic sedimentation laying the retro-arc foreland basin (Romans et al., 2011).

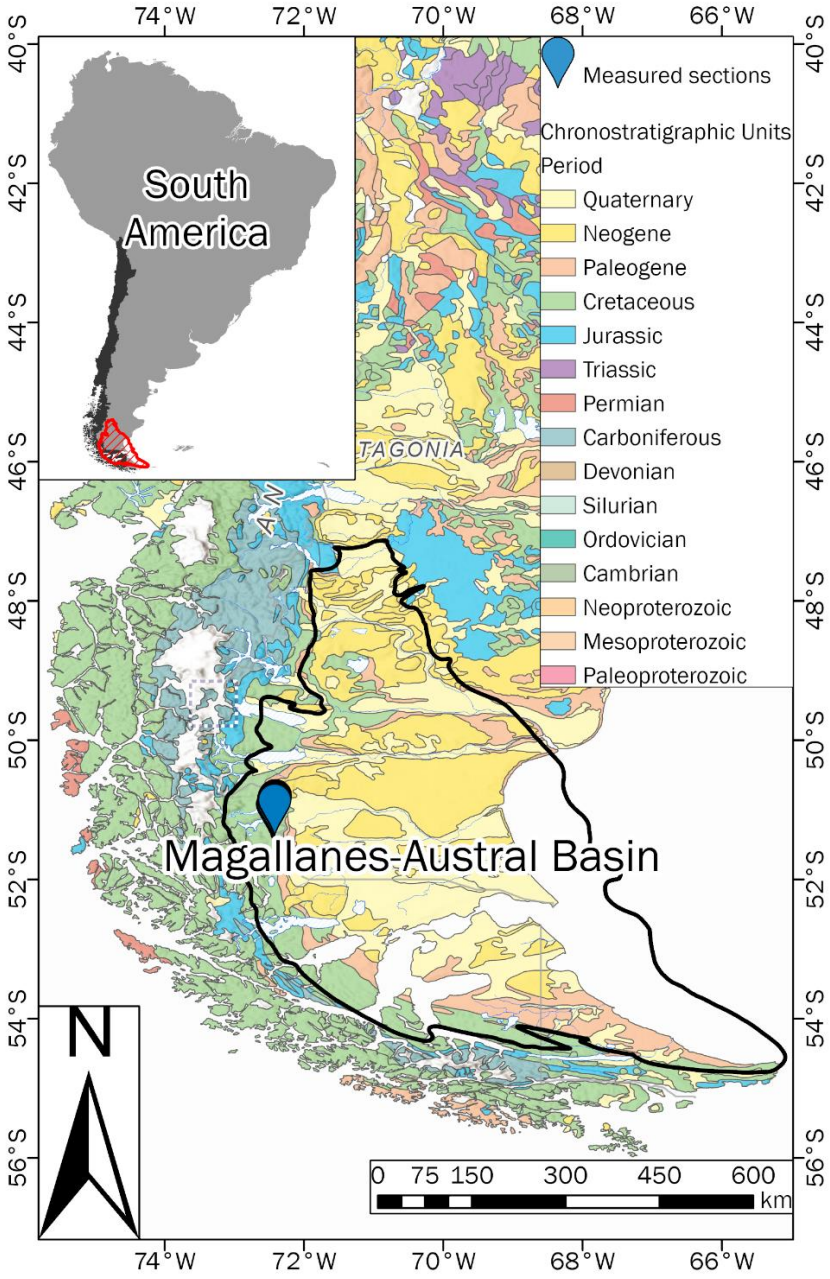


Fig. 2. Geology of the Magallanes (Chile) or Austral (Argentina) Basin (after Gómez et al. (2019)). The Tres Pasos Fm. occurs as part of the western Cretaceous thrust belt adjacent to the Neogene foreland basin. The approximate location of the measured sections is also shown.

The Cerro Toro Fm. (86 – 80 Ma) comprises conglomerate-filled turbidite channel-levee systems within the overall shale-dominated succession and occurs as stacked conglomeratic and sandstone channel fills with associated finer-grained overbank deposits. The thickness of this formation is 2500 m, deposited in a foredeep-axial channel-levee system. This formation lays concordantly above the Punta Barrosa Fm. (Mpodozis et al., 2011; Romans et al., 2011).

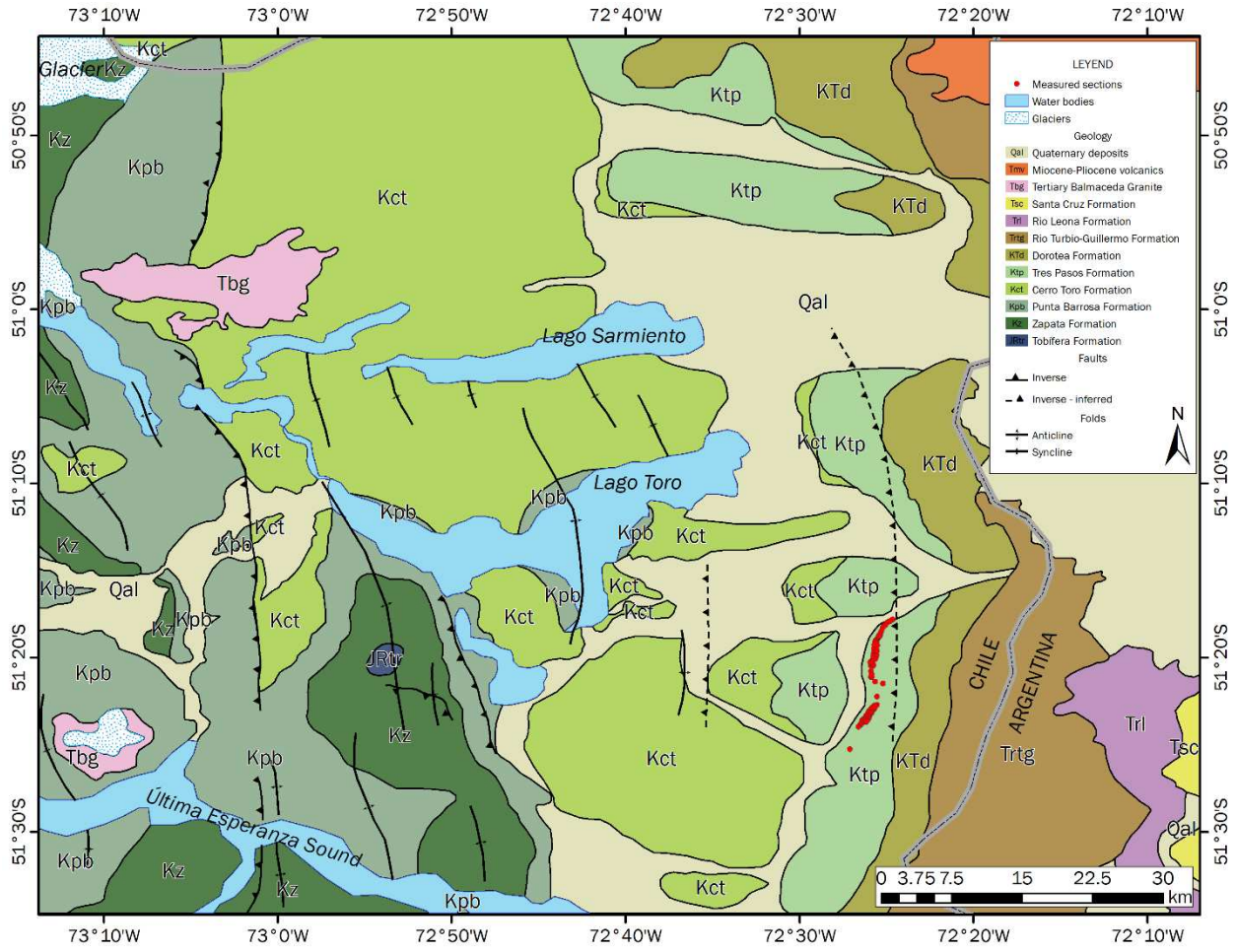


Fig. 3. Geology of the Ultima Esperanza District, southern Chile, showing the lithostratigraphic units of the Magallanes Basin. Formations are younger and less structurally deformed to the east (Austral Basin, Argentina). The location of the measured sections is shown in red points. After Fosdick et al. (2011).

The Tres Pasos Fm. (80 - 70 Ma) consists of sandstone-rich successions and mudstone-rich mass transport deposits (MTDs) with highly variable thickness (from 1200 to 1500 m) and cross-sectional geometry, and occurs as a basal mass transport deposit at the base of the slope and lower slope segments and as prograding delta-fed slope clinoforms systems over the top (Mpodozis et al., 2011; Romans et al., 2011).

Finally, the Dorotea Fm. (70 – 68 Ma) composes massive glauconitic sandstones with conglomerate intercalations showing topset strata related to shallow-marine and deltaic deposits (Mpodozis et al., 2011). This formation is genetically linked to the Tres Pasos Fm., in the sense that has been interpreted that it prograded along the basin axis to the south (Romans et al., 2011). The stratigraphic column of the Magallanes Basin is shown in Fig. 4.

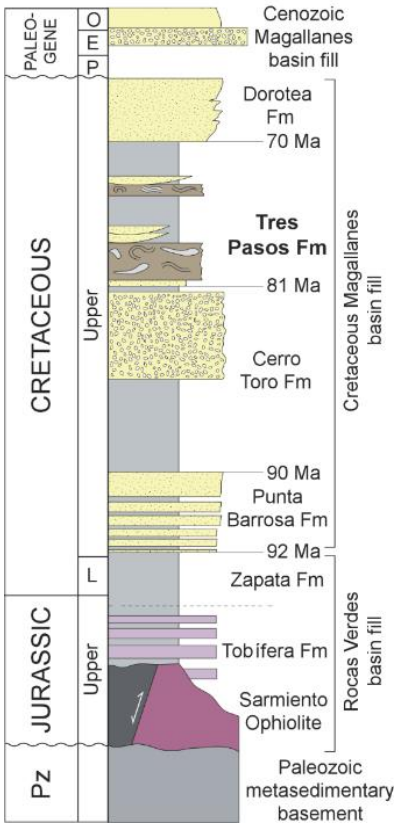


Fig. 4. Stratigraphic column of the Magallanes Basin in the study area. From Daniels et al. (2018).

2.2. Deep-water or submarine channels

A *deep-water* or *submarine channel* is a negative topographical feature produced mainly by turbidity currents that transport sediment and can be sites of deposition or erosion (Fig. 5). Channels can be classified on the degree of confinement of the channel, which systematically changes downflow. Highly confined channels form when the flow is contained within the channel and include submarine canyons and erosional channels. In contrast, poorly confined channels with levees form when flows can escape the channel confinement to build channel-bounding levees (Arnott, 2010).

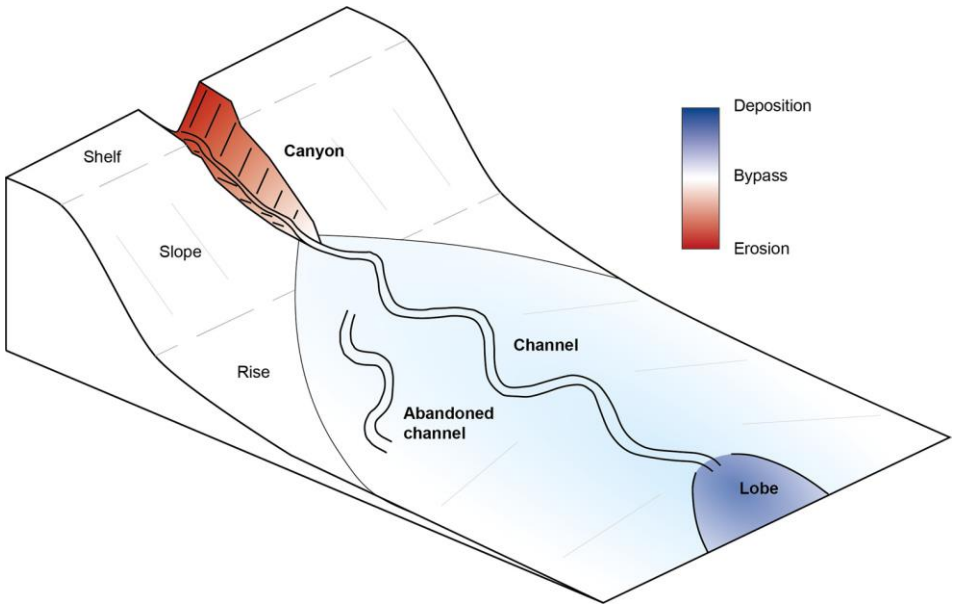


Fig. 5. Schematic morphology of submarine or deep-water depositional environments, showing in colors how erosion and deposition are relatively distributed over space. The upstream canyon is characterized by erosion, the channel by erosion, bypass or deposition, and the lobe is the main depositional feature (Heijnen et al., 2022).

2.2.1. Hierarchy of submarine channels

In deep-water or submarine channelized systems, the smallest stratigraphic mappable architectural unit is the *channel element*, which consists of a channel-form surface and its sediments (Fig. 6A, B). Different channel elements are characterized by an abrupt vertical or lateral offset of channel element

architectural position or facies. Multiple channel elements that stack in a consistent pattern compose a *complex*. If multiple complexes are present, they constitute a *complex set*. The term *system* is most often equivalent to one or more complex sets (McHargue et al., 2011b).

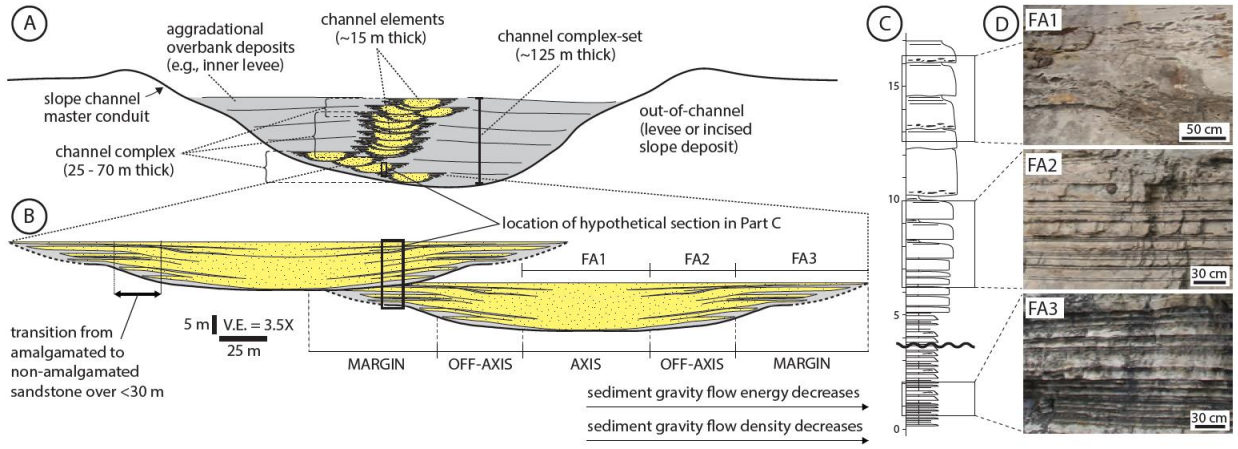


Fig. 6. (A) Overview of the stratigraphic hierarchy classification scheme utilized for channel strata at Laguna Figueroa (B) Elements are composite features, which represent the incision and filling of individual channels. Similar sedimentation units are grouped into sedimentation unit associations or facies that can be related to intra-element architectural positions: axis, off-axis, and margin. (C) Schematic measured section through one channel element, with the dark wavy line demarcating an erosional contact. (D) Photographs of sedimentation unit associations or facies. They represent the fundamental architectural division (Hubbard et al., 2023; Macauley and Hubbard, 2013).

2.2.2. Submarine channel element architectural positions

Submarine channels exposed in the Tres Pasos Fm. (Magallanes Basin, Chile) are low-sinuosity channel elements with symmetric to slightly asymmetric fill. An erosive contact limits the channel elements to the top, and a laterally extensive (but not always continuous) base drape outlines the channel element to the base (Hubbard et al., 2018). The Tres Pasos Fm. was initially interpreted to contain eighteen channel elements, grouped into three channel complexes (Fig. 6) (Macauley and Hubbard, 2013). Recent reinterpretations point out 25 stacked channel elements, twelve of them belonging to the Lower Laguna Figueroa Complex set (Ruetten, 2021; Southern et al., 2017; Vento, 2020). That complex set is the focus of this thesis, and the code was written regarding the measured section CACH1 (Fig. 7).

These channels display three channel element architectural positions: *axis*, *off-axis* and *margin* (Fig. 6C, D) (Macauley and Hubbard, 2013). Mostly thick-bedded, highly amalgamated sandstones *sedimentation unit associations or facies* constitute the *axis architectural position*. Mostly thick- to thin-bedded semi-amalgamated sandstones intercalated with mudstones facies compose the *off-axis architectural position*. Finally, intercalations of thin-bedded non-amalgamated sandstones and mudstones facies comprise the *margin architectural position* (Covault et al., 2016; Macauley and Hubbard, 2013; McHargue et al., 2011b, 2011a). The transition from axis to margin architectural position usually occurs across less than 30 m (Macauley and Hubbard, 2013).

2.3. Previous modeling work

This research is part of a series of master's projects focused on the modeling of the lowermost Laguna Figueroa channel complex outcrops (Fig. 7). Those projects have been conducted by Dr. Lisa Stright's students at the Department of Geosciences at CSU and built upon the previous work of researchers in the Chile Slope Systems joint industry project.

Sedimentological observations and interpretations of the Laguna Figueroa outcrop (Fletcher, 2013; Macauley and Hubbard, 2013) are the foundation for a database of outcrop statistics (https://www.chileslopesystems.com/) that has been used to predict stratigraphic architecture (i.e., channel element architectural positions) in deep-water slope channel system, addressing the gap between sedimentology and data analytics using machine learning techniques (Vento, 2020).

Ruetten (2021) created a deterministic outcrop model that is 265 m height x 2.25 km long (oriented north to south) x 2 km wide (oriented east to west), with ~ 5.7 M grid cells that are 50 m x 50 m x 2.5 m. The model includes two channel complex sets (Lower and Upper Figueroa) separated by a mass transport deposit (MTD) (Ruetten, 2021) associated with complex and complex set surfaces. Within the lowermost channel complex set, 12 channel elements were revisited from the original interpretation of 18 channel elements (Fig. 7) (Ruetten, 2021; Southern et al., 2017; Vento, 2020).

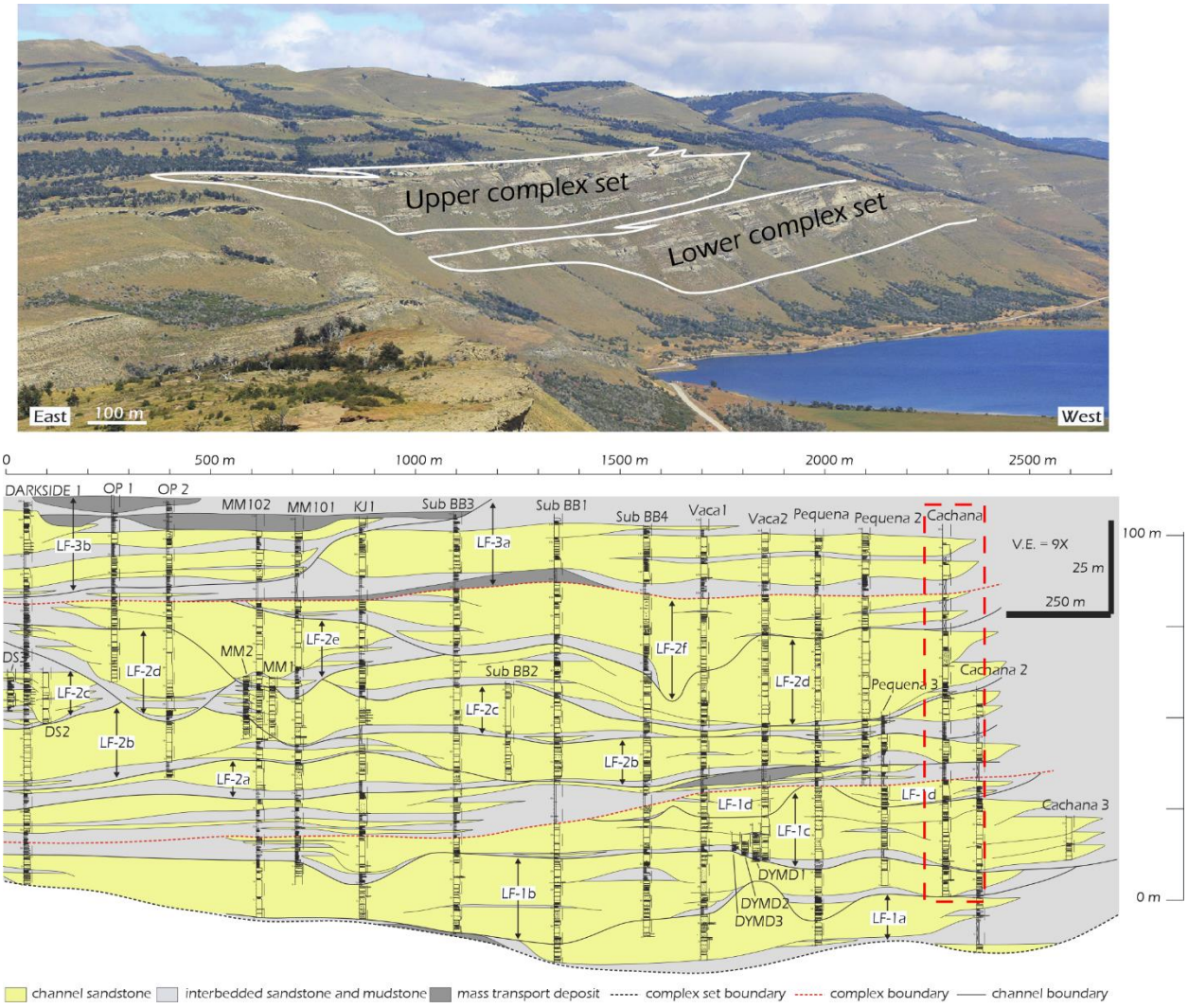


Fig. 7. Upper panel, photo of the upper and lower channel complex sets at Laguna Figueroa with complex sets outlined. Lower panel, an oblique dip-oriented cross-section of Lower Figueroa with channel complex sets and complexes (Vento, 2020). The measured section CACH1 is highlighted by a red rectangle.

The channel element template used has a constant width of 400 m and is 25 m thick (Ruetten, 2021). Previous channel element dimensions used for modeling efforts from Dr. Stright's research group were 200-300 m wide x 14 m thick (Jackson et al., 2019; Meirovitz et al., 2020).

That study concluded that drapes and/or low net-to-gross (NTG) margins acted as baffles that reduced water breakthrough. Increasing the drape leads the baffle to become a barrier. Furthermore, reservoir compartmentalization is reported to be created by a variety of factors, like channel element base

drape coverage, laterally divergent stacking patterns, low NTG margins, and the presence of mass transport deposits (MTDs) (Ruetten, 2021). Channel stacking pattern is again highlighted as an important factor that impacts connectivity (Jackson et al., 2019; Meirovitz et al., 2020).

While these previous studies reveal the critical importance of channel element stacking patterns to fluid flow and connectivity, channel elements are below seismic resolution and predicting their presence and organization is difficult to achieve interpretively. Channel element stacking at a well location is directly linked to the architectural element positions (axis, off-axis, and margin) that the wellbore intersects. Machine learning algorithms were used to classify variables that predicted the probability of channel element architectural positions (Vento, 2020). Complex techniques (random forests, XGBoost, and neural networks) had higher accuracies (above 80%), while relatively less complex techniques (decision trees) had lower accuracies (between 60% - 70%). The transitional off-axis architectural position was reported to be the most difficult to classify by machine learning techniques (Vento, 2020). However, classifying the architectural element positions at the wellbore requires an extra step of modeling channel element stacking in 2D and 3D. Many studies have addressed different aspects of deep-water channel modeling.

CHAPTER 3. PREVIOUS WORK ON MODELING CHANNEL STACKING PATTERNS

3.1. Previous studies on submarine channel stacking patterns

Several authors have provided insights into how submarine channels are deposited and stacked using (i) statistics from outcrops, wells, seismic data and modern analogs with event-based computational models (McHargue et al., 2011b, 2011a), (ii) perturbation methods in grids/rasters (Li and Caers, 2011), (iii) observations from outcrops (Macauley and Hubbard, 2013), (iv) Non-Uniform Rational B-Splines (NURBS) with sequential Gaussian simulations (SGS) or multiple-point simulations (MPS) (Rongier et al., 2017), and (v) channel trajectories (Morris et al., 2022; Sylvester et al., 2011).

McHargue et al. (2011a, 2011b) stress the importance of event-based models of submarine channels given the lack of deep-water system outcrops expose on the Earth's surface to provide accurate constraints on the channel stacking patterns (2D) or even the geobodies (3D). Event-based models are constrained by *rules*. A rule is "an observed pattern or trend, a statistically defined relationship, or even an expert opinion" (McHargue et al., 2011a), and rules can be either *empirical* or *predictive*. *Empirical rules* are quantitative summaries of trends, patterns, or dimensions with defined distribution and uncertainty and can be measured directly. In contrast, *predictive rules* are hypotheses that build on incomplete knowledge of processes. Predictive rules require inference of fundamental processes of sediment transport and deposition, meaning that they are more linked to geological thinking. Some predictive rules used in those event-based models are:

- Assuming that channel gradient is constant across the length of a hydrocarbon reservoir.
- Allogenic cyclicity is a simplifying assumption imposed by varying model parameters. Autogenic processes are modeled through the interplay of multiple rules.
- Channel elements tend to stack with lateral offsets of less than a channel width in channel complexes with organized channel stacking.

- Channel systems increase aggradation rate with time and evolve from disorganized stacking to organized stacking patterns.
- The channel stacking of channel elements deposited within a channel complex set results in a succession of channel complexes that tend to follow a pattern.

In that study, a *channel stacking pattern* is defined as the placement of one channel element relative to another, and two kinds of channel stacking patterns were recognized: *disorganized* and *organized* (Fig. 8).

Organized channel stacking patterns occur when one channel element influences the deposition, and therefore the location, geometry and stratigraphic architecture of the next channel element. In the end, the two channel elements are very similar, providing a useful basis for predicting features of successive channel elements within the same channel complex. *Disorganized channel stacking patterns* occur when one channel element has little or no influence on the location, geometry and stratigraphic architecture of the next channel element, so the features of successive channel elements cannot be predicted. Sketches of the planform of organized vs. disorganized channel stacking patterns are shown in Fig. 8. The concepts of organized vs. disorganized channel stacking patterns and some of the predictive rules have been widely used in other studies and are useful for comparison purposes.

Li and Caers (2011) illustrate a channel stacking pattern modeling technique that uses a distribution function of pattern parameters to simulate and perturb channel stacking patterns. Instead of simulating lateral and vertical offsets, they used the *overlap* and *migration ratios*. The *overlap ratio* is defined as the ratio of vertical overlap (h) between two channels and the channels' maximum thickness (H). The *migration ratio* is the ratio of horizontal distance (x) between two adjacent channel centerpoints and the channel amplitude (A) (Fig. 9). The probability distribution functions (PDFs) of the overlap and migration ratios could be obtained from outcrops, training images or event- or process-based models. Channel element cross-sections are defined by a parabola, and perturbations are done as a series of operations with grids/rasters in a pixel-based stochastic engine.

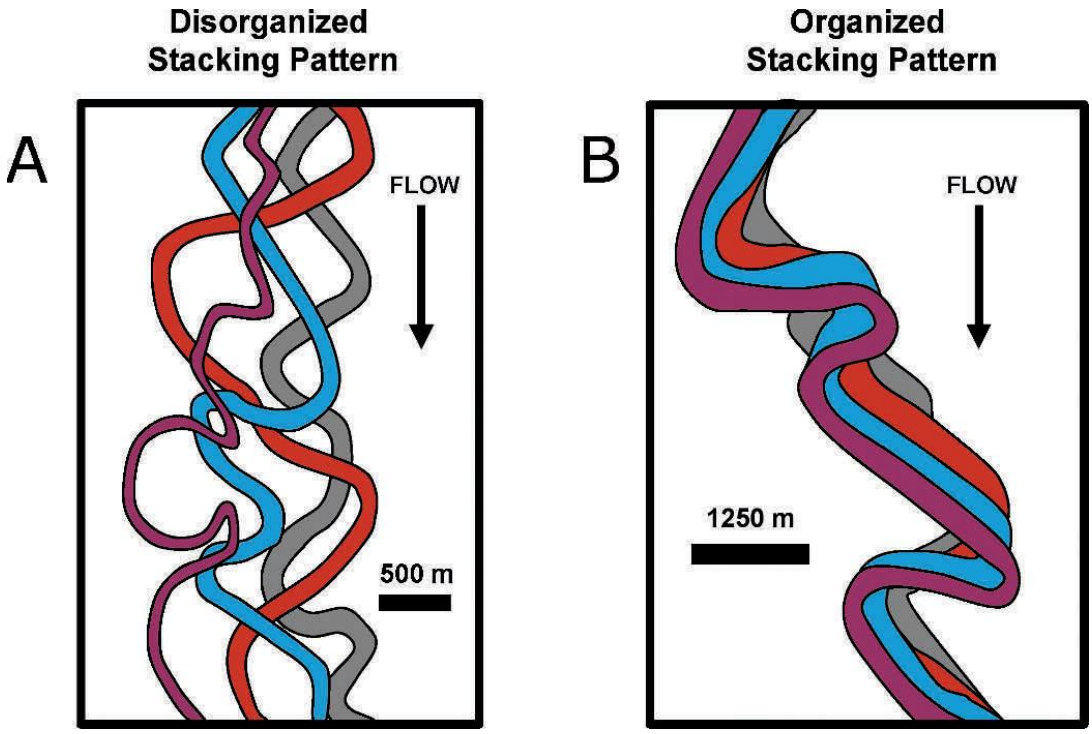


Fig. 8. Planform of channel stacking patterns. A) Disorganized stacking pattern. B) Organized stacking pattern. The location, geometry and stratigraphic architecture of successive channel elements are similar to the first channel element for B (McHargue et al., 2011a).

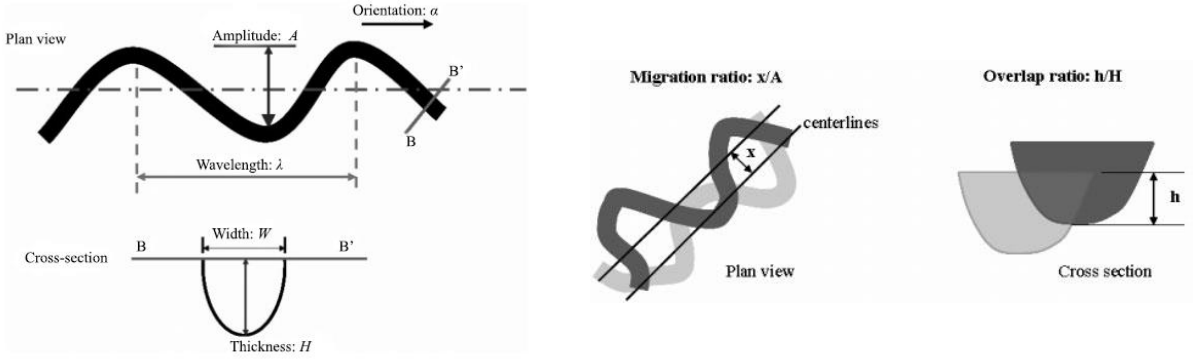


Fig. 9. Parameters used to describe the channel element geometry and to perturb or model the channel stacking patterns (Li and Caers, 2011).

The authors highlight how challenging the modeling of individual channel element geometry, channel stacking patterns and well data conditioning or matching are, and also express the need for well or measured section data to be interpreted in terms of channel element architectural positions.

Using the Tres Pasos Fm. outcrops (Magallanes Basin, Chile), Macauley and Hubbard (2013) used the definitions of channel elements, channel complexes and channel complex sets and interpreted 18 channel elements and three channel complexes. Furthermore, with detailed sedimentological and stratigraphical observations and paleocurrents, they followed and drew the approximate planform geometry of the 18 channel elements. Digitizing this manual interpretation, they also created five depositional strike cross-oriented sections displaying the channel stacking patterns throughout the three channel complexes (Fig. 10). Using concepts like lateral and vertical offset for describing how the vertex of the channelized form in an x, y plane changes throughout the channel elements, they concluded that greater than 80% of all channels stacked within a half-width of the previous channel, demonstrating the limited lateral offsets observed in that formation.

A geometrical and descriptive approach to stochastically model the channel stacking patterns is developed by Rongier et al. (2017) relying on the simulation of an initial channel using a fractals generator (Lindenmayer system) and simulating a migration factor using a sequential Gaussian simulation (SGS) or a multiple-point simulation (MPS). The main difference between this modeling approach from the ones of McHargue et al. (2011a, 2011b) is that this geostatistical simulation reproduces the geometry and stratigraphy (distribution in time and space) resulting from the physical processes rather than modeling the physical processes themselves. However, measured section or well data conditioning is not explored.

Sylvester et al. (2011) and Morris et al. (2022) use channel trajectories. Their models, besides accounting for lateral offset or channel migration, also account for parameters such as incision, aggradation, channel and overbank deposits and cutoffs, and use the following predictive rules or assumptions:

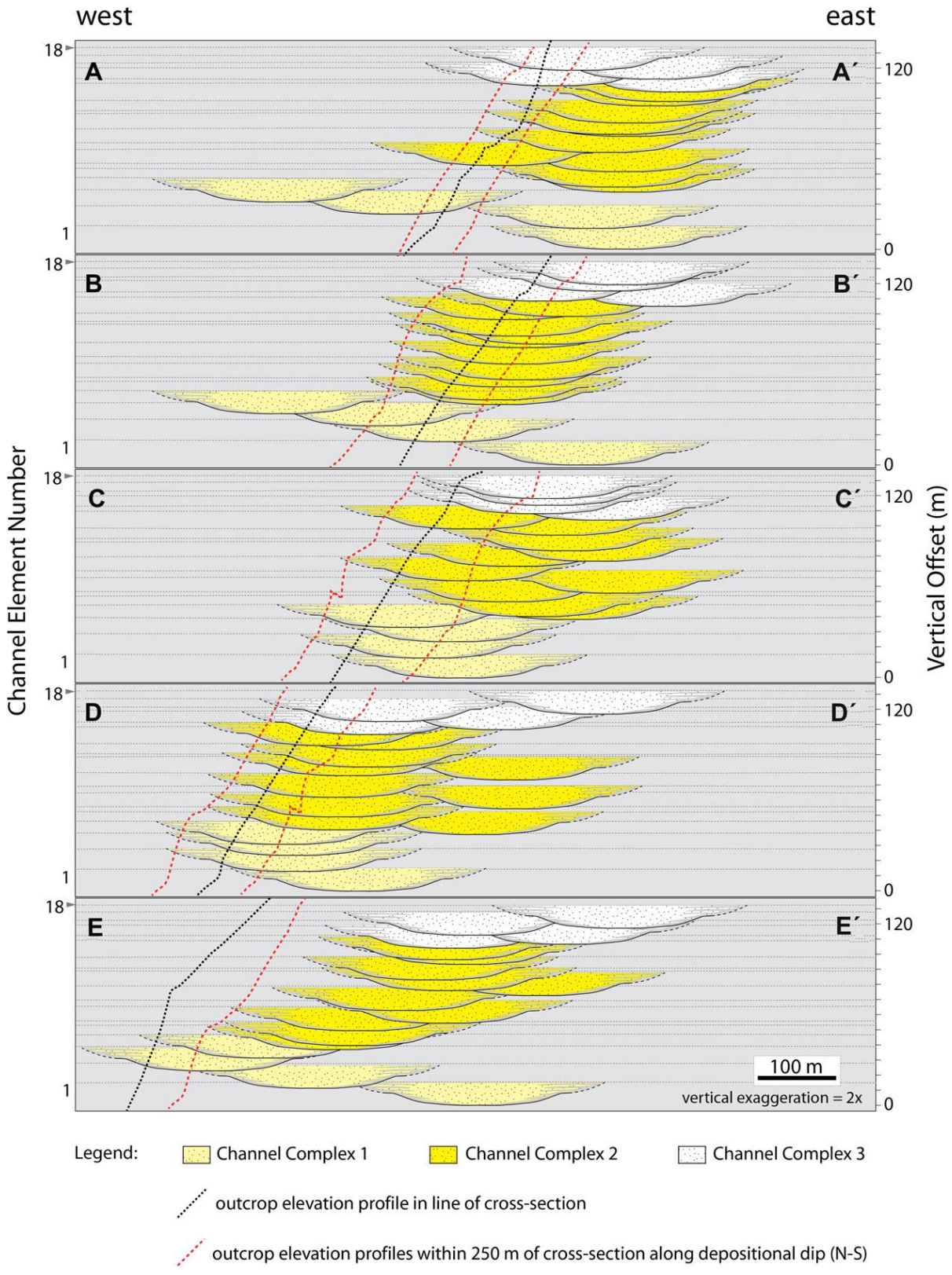


Fig. 10. Depositional-strike-oriented cross-sections. Vertical and lateral offset is measured between successive channel elements to quantitatively analyze stacking patterns (Macauley and Hubbard, 2013).

- Within individual submarine channel systems, channel size variability is relatively small, meaning that channel elements within a channel complex tend to have a characteristic and relatively constant size and shape.
- Channels migrate systematically and their placement is not random in organized channel stacking.

The modeled channel stacking patterns are created by modifying the migration, or *total offset* of a single channel element or shape, with varying degrees of *vertical offset* or aggradation and *lateral offset* or lateral migration. Furthermore, they highlight that some seismic studies suggest that the *vertical offset* or component of the channel trajectory is unlikely to change sign (eroding and depositing) frequently, and often consists of a single incision-aggradation, with no evidence for multiple reincisions. They also used a parabola for the parameterization of the channel elements. The channel centerpoint is the channel midpoint at the top, and at each time step, the new location of the centerpoint is defined by a lateral offset or horizontal component and a vertical offset or component applied to the coordinate at the previous time step. Finally, the parameters x and z define the channel trajectory. Their channel element template is 600 m in width x 60 m thick, in agreement with the work of Jobe et al. (2016).

Furthermore, Morris et al. (2022) use a centerline creation algorithm designed for fluvial models (e.g., Pyrcz et al., 2009) to generate deep-water channels (Covault et al., 2016), then hypothesizing that the stratigraphic complexity of submarine channels can be explained or reproduced by a continuous flow (Sylvester et al., 2011) instead of by punctuated events (Arnott, 2010; Hubbard et al., 2014; Macauley and Hubbard, 2013).

3.2. Differences from previous work

Macauley and Hubbard (2013) provide fundamental concepts for organized channels from outcrops. Nevertheless, the planform geometries of the channels from which the cross-sections of the channels stacking patterns were created should be considered as an initial, deterministic, qualitative

approximation. Our approach for generating the channel stacking patterns is based on machine learning-derived probabilities based on the new interpretation (Vento, 2020). The incorporation of probabilities results in a range of stochastic models that all match the measured section data, rather than a single interpretation.

Additional CSS internal work has lumped and reinterpreted channel elements, reducing the total number of elements from 18 to 12 in Lower Laguna Figueroa (the focus of this study). Therefore, the channel elements are 400 m width x 25 m thick, rather than the 300 m width x 15 m thickness of Macauley and Hubbard (2013). These values are consistent with the most recent modeling studies of channel elements in the Tres Pasos Fm. (Langenkamp, 2021; Ruetten, 2021). Exact measurements of channel element width and height are difficult to ascertain from subsurface data, and outcrop and modern analog studies present a range of values (McHargue et al., 2011a). Therefore, channel width and thickness (treated as constant in this study) can be treated as a variable in future modeling approaches.

Sylvester et al. (2011) use a similar approach to ours, using parabolas as a template for channel elements, and calculating a total offset as a result of two components, the vertical and lateral offsets. However, instead of defining the channel centerpoint as the channel midpoint at the top, our channel element centerpoint is the parabola vertex. Channel trajectories are better to conceptualize and quantify channel evolution in a slope-parallel section, whereas our approach is suitable when measured section or well data conditioning and matching is required.

3.3. Individual channel element geometry parameters

For representing and modeling channelized forms, parabolas have been extensively used (Li and Caers, 2011; Morris et al., 2022; Rongier et al., 2017; Sylvester et al., 2011). The equation used in this study is below (Eq. 1), and parameters like width (w), thickness or depth (t) and height (k), lateral offset (x) and vertical offset (y) were used for modeling the channel stacking patterns (Fig. 11).

$$Parabola\ (p) = ax^2 + y = \frac{t}{\left(\frac{w}{2}\right)^2} \times\ (x_0 - x)^2 + y \quad (1)$$

3.3.1. Width (w)

Width is defined as "the maximum width of the channel; it is located at the top of the channel" (Li and Caers, 2011) without erosion. The widths of channel elements fall between 100 and 800 m (outcrop)/600 m (subsurface) with an average of 300 m (outcrop)/200 m (subsurface) (McHargue et al., 2011b). Measurements from outcrops suggest a wider range of values with an average width of 300 m. Widths measured from outcrops are apparent widths (McHargue et al., 2011b). As mentioned above, for our channel element template we used a width of 400 m (Langenkamp, 2021; Ruetten, 2021).

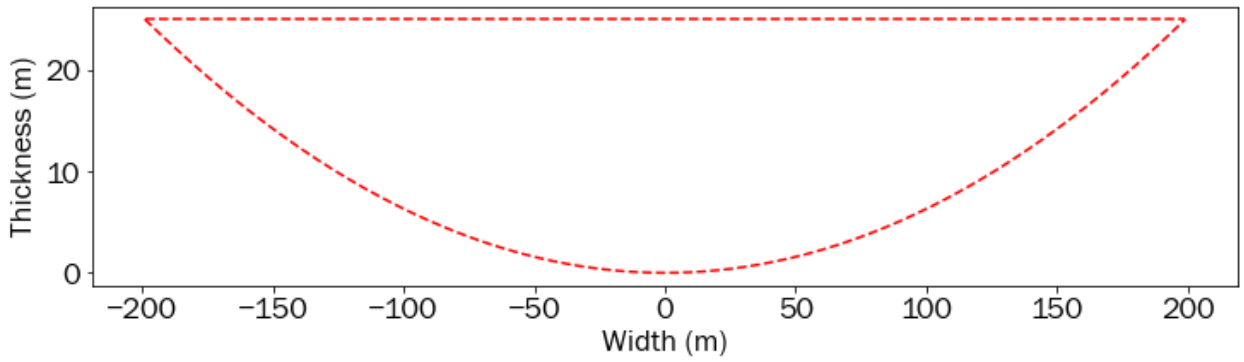


Fig. 11. Template used in this study. We used a template of 400 m width x 25 m thick.

3.3.2. Thickness or depth (t) and height (k)

Thickness is "the maximum thickness of the channel; it is located at the center of the channel" (Li and Caers, 2011). Preserved element thickness values are on average 10.7 m (subsurface)/13 m (outcrop) with maximum observed values between 35 and 40 m (McHargue et al., 2011b). The thickness used for this research was 25 m (Langenkamp, 2021; Ruetten, 2021) based on observed outcrop data. Thickness is

inversely related to height in a parabola (i.e., when the parabola reaches the maximum thickness, height is 0). Thickness is defined and measured as a perpendicular line that goes from the vertex to the middle top of the parabola. In contrast, height is defined by a perpendicular line from an x-plane at the vertex of a parabola to the base of the parabola.

3.3.3. Lateral offset (x)

Lateral offset is defined as the x-component of an *x, y* point that defines the vertex of a parabola. This concept is introduced by Macauley and Hubbard (2013) (Fig. 12).

3.3.4. Vertical offset (y)

Vertical offset is defined as the y-component of an *x, y* point that defines the vertex of a parabola. This concept is introduced by Macauley and Hubbard (2013) (Fig. 12).

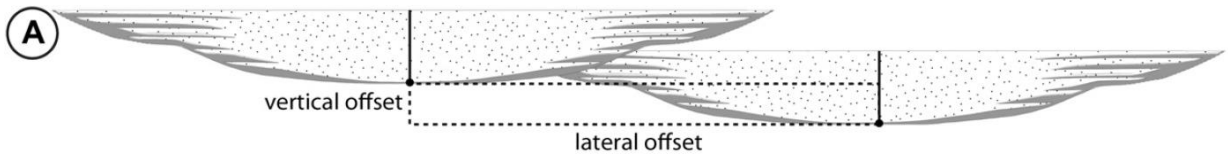


Fig. 12. Sketch of two stacked channel elements to illustrate measurements of vertical and lateral offset (Macauley and Hubbard, 2013). Notice that both components are measured regarding the vertex of the channelized from, in contrast to Sylvester et al. (2011).

3.4. Methodology workflow

Here we present the general pseudocode that we used to guide the programming in Python (Fig. 13):

1. Read axis/off-axis/margin and thickness for each channel element probabilities (Neural Network [NN], Vento (2020), Chapter 4; Transition Probabilities [TP]; Chapter 5). Read A/OA/M and thickness for

each channel element probabilities (Neural Network [NN], Vento (2020); Transition Probabilities [TP]; Chapter 5)

2. Convert PDF to CDF
3. For s = number of realizations
    a. Conditional simulation (n=1, first channel element): random draw from CDF for axis/off-axis/margin
        i. NN conditioning to well, Chapter 4 or uniform random draw as start for TP, Chapter 5
        ii. If off-axis or margin drawn, random uniform draw to right off-axis/margin or left off-axis/margin, if axis drawn, then axis
        iii. **Result:** element n, position axis / (left/right) off-axis / (left/right) margin for element n = 1
    b. Conditional simulation (For n=2 to num elements); random draw from CDF for axis/off-axis/margin
        i. CDF:
            1. NN results at channel element = n (Chapter 4);
            2. TP based on random draw for n = 1 (Chapter 5)
        ii. If off-axis or margin drawn, random draw to right off-axis/margin or left off-axis/margin, if axis drawn, then axis
        iii. Generate parabola and plot data
    c. **Result:** element n=2 to num elements, position axis / (left/right) off-axis / (left/right) margin; i.e., multiple equiprobable realizations that are conditioned to probabilities at the wellbore (Chapter 4) or conditioned to honor TP (Chapter 5)
    d. For n = 1 to number of channel elements
        i. Shift channel element vertical position to match thickness in 1D data (Chapter 6)
        ii. Generate parabola and plot data

iii. **Result:** multiple equiprobable realizations that are conditioned to probabilities at the wellbore (Chapter 4) or conditioned to honor TP (Chapter 5) and place correctly to match thicknesses at the wellbore

e. Break when the number of unique possibilities is reached.

We also performed a sensitivity analysis for:

a. The number of simulations required to generate all equiprobable cases and

b. Evaluate the 10 most likely and 10 least likely cases.

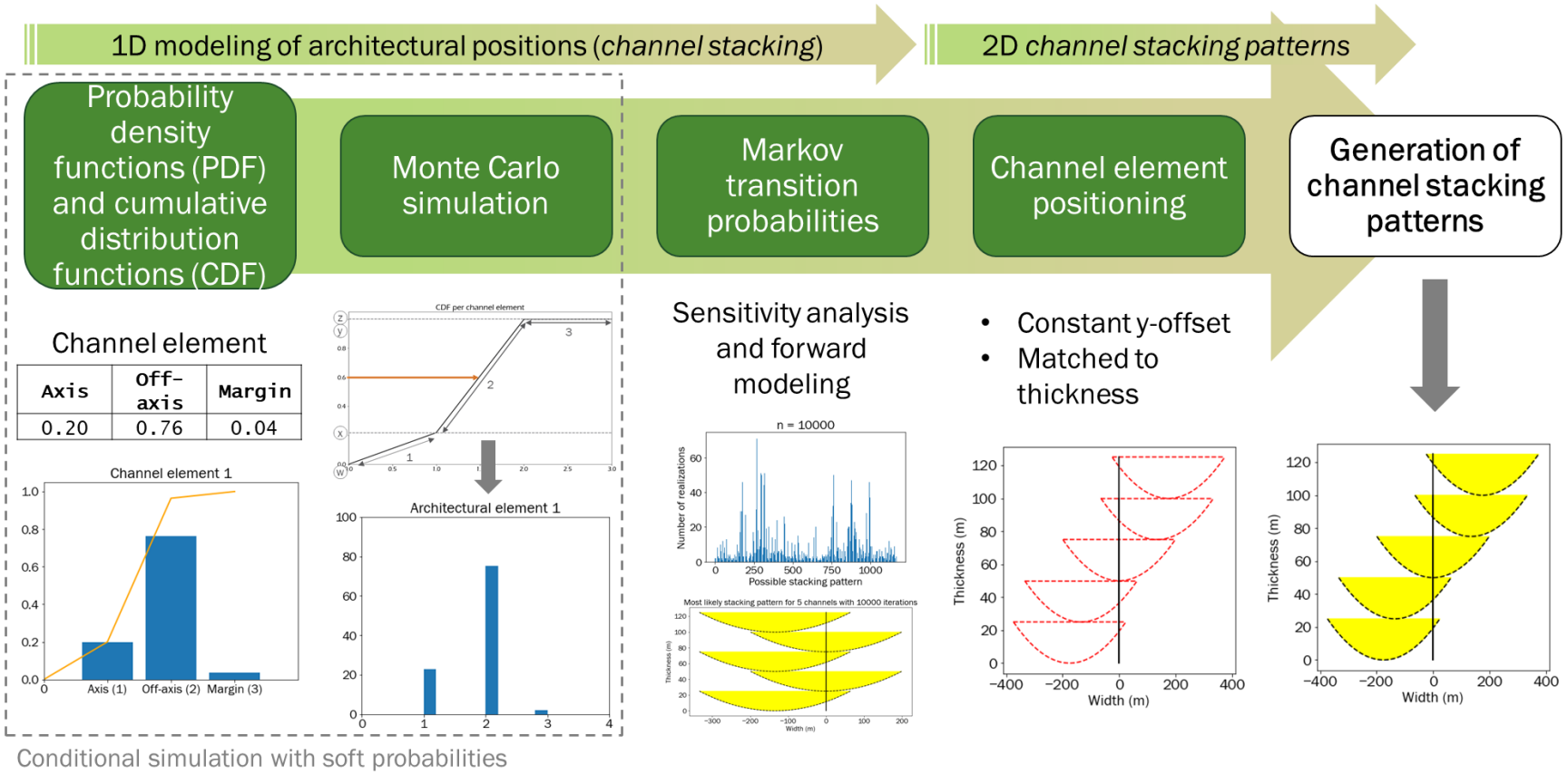


Fig. 13. Overview of the methods used in this research.

CHAPTER 4. MODELING CHANNEL STACKING PATTERNS WITH CONDITIONAL SIMULATION AND SOFT PROBABILITIES

4.1. Motivation

Interpretation of stacking patterns has been qualitative and deterministic (Macauley and Hubbard, 2013). With the advent of new technologies for reprocessing vast quantities of data in a short time, machine learning techniques were used on several measured sections of the Tres Pasos Fm. A neural network algorithm classified the channel element architectural positions (Vento, 2020). Assuming the probabilities of classification as probabilities of occurrence, it is possible to use them to model stochastically deep-water channel stacking patterns for the first time.

A computational straightforward procedure for doing it is Monte Carlo simulation. Here, we try to get equiprobable realizations modeling individual channel elements and modeling channel stacking (i.e., how architectural positions stack vertically) using conditional simulation with the soft probabilities of occurrence and extrapolate those results to get 2D representations of the channel stacking patterns.

4.2. Methods

4.2.1. Database and coding style

For performing the modeling of the channel stacking patterns with independent probabilities, we used machine learning-derived probabilities from the neural network classification of architectural positions of Vento (2020). Those probabilities are condensed in the .xlsx file NN_Results.xlsx (not included in this thesis). In this thesis, we include a modified version of that file, that includes the thicknesses of each channel element per architectural position (NN_Results_2.xlsx, Appendix A).

For each channel element, there is an axis, off-axis and margin architectural position probability of interpretation or occurrence. Those probabilities then sum up 1, or 100%. From Vento (2020), only some

channel element architectural positions are available per measured section, due to the occurrence of mass transport deposits (MTDs), stratigraphic intervals covered by vegetation or debris and intervals that were not classified. We are assuming then a measured section without gaps, with a complete record and without stratigraphic vertical complexity (no mass transport deposits – MTDs occurrence).

Then, we proceeded to upload this .xlsx file as a pandas.DataFrame in Python using JupyterLab. The scripts were written in a functional programming style (compare with object programming style).

4.2.2. Random variables

The fundamental concept of all statistics and probability theory is that of a random variable or function. A random variable is "a variable that can take a series of possible outcomes, each with a certain probability or frequency of occurrence" and is denoted by the letter Z (Remy et al., 2009). In contrast, a deterministic variable takes only one value or outcome. A random variable can be either discrete or continuous. For discrete random variables, each outcome is attached to a probability value. For a continuous random variable, the distribution of probability values can take the form of a cumulative distribution function (CDF) or a probability density function (PDF). Even though lithological classifications are typically expressed as a discrete random variable, expressing their probabilities in a histogram resembling a probability density function (PDF) and then creating a cumulative distribution function (CDF) provides us with tools to model them stochastically.

4.2.3. Cumulative probability functions (CDFs)

A cumulative distribution function (CDF) provides the probability for a random variable not to exceed a given threshold value and it is pictured as a cumulative histogram (Remy et al., 2009) (Fig. 14a).

4.2.4. Probability density functions (PDFs)

A probability density function (PDF) is defined as the derivative or the slope of the previous cumulative distribution function (CDF) at values where the CDF is differentiable, and is pictured as a histogram (Remy et al., 2009) (Fig. 14b).

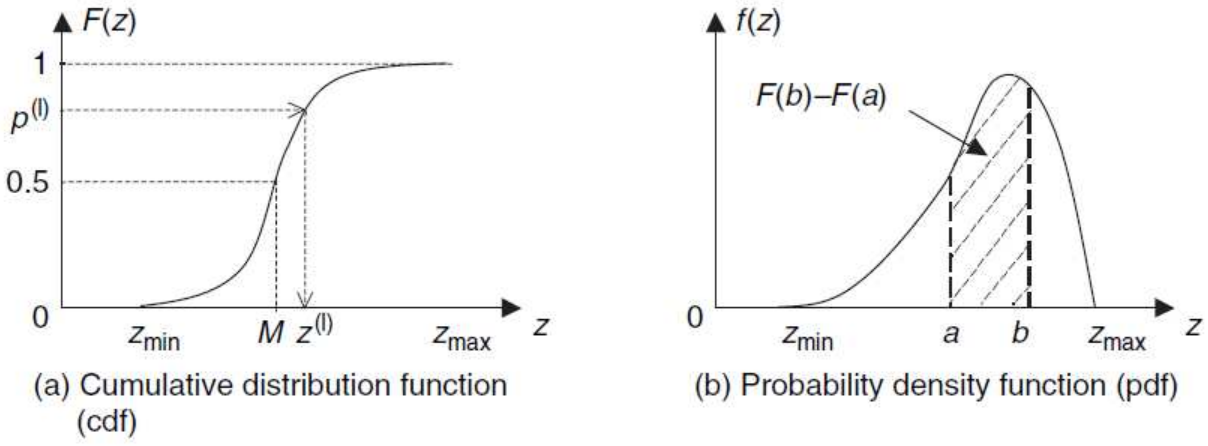


Fig. 14. Sketches of (a) a cumulative distribution function (CDF) and (b) a probability density function (PDF) (Remy et al., 2009).

4.2.5. Drawing probability density functions (PDFs) and cumulative distribution functions (CDFs) in Python

After uploading the curated .xlsx file into JupyterLab as a pandas.DataFrame, we selected the measured section and the axis, off-axis and margin architectural position probabilities and converted them into a NumPy array using the function *values*.

Then we defined two functions, one for generating NumPy arrays for drawing the probability density functions (PDFs) or plotting as bars the architectural positions and another one for generating NumPy arrays for drawing the cumulative distribution functions (CDFs).

The PDFs arrays generator function (*prob* function) adds a row of 0 to the NumPy array with the architectural positions' probabilities, so we could draw the CDFs on top and compare. The CDFs arrays

generator function (*prob_cumsum* function) calculates the cumulative sum of the architectural positions probabilities and then inserts a 0 value as the first element to draw the CDFs from (0,0) to (3,1).

An architectural element position names array (ArchitecturalElementPositionNames) was created to plot both the PDFs and the CDFs. Then, the *pdfs_and_cdfs* function plots the PDFs and CDFs distribution of the architectural position probabilities per channel element within the measured section. This function plots the PDFs and the CDFs from the geological or stratigraphical base to the top and labels them correspondingly.

4.2.6. Monte Carlo simulation

Statistical simulation is a popular method to address problems that require the stochastic generation of multiple scenarios to assess uncertainty. One of those methodologies is to draw samples that follow a desired probability density function (PDF). Once many realizations have been obtained, the final PDF function is estimated from those samples. This procedure is termed Monte Carlo simulation, and some practical applications for geological modeling are provided by Avseth et al. (2005).

For the simple case of a univariate variable and a known PDF, we can perform a Monte Carlo simulation by drawing n samples by first generating a uniform random value between 0 and 1 and then evaluating the results at the CDF, which is a monotonic increasing function and take values between 0 and 1. Hence the simulated values follow the desired PDF.

Univariate Monte Carlo simulation may be described as shown in Fig. 15.

After extending the data by Monte Carlo simulation, it is important to check that the estimated and original values have similar statistical distributions. This can be done by plotting comparative histograms, quantile–quantile plots and scatter plots of the original and the simulated values (Avseth et al., 2005). For performing this check, we compare visually the PDFs, or bar plots of the architectural position probabilities with the histograms of the distribution of the samples simulated.

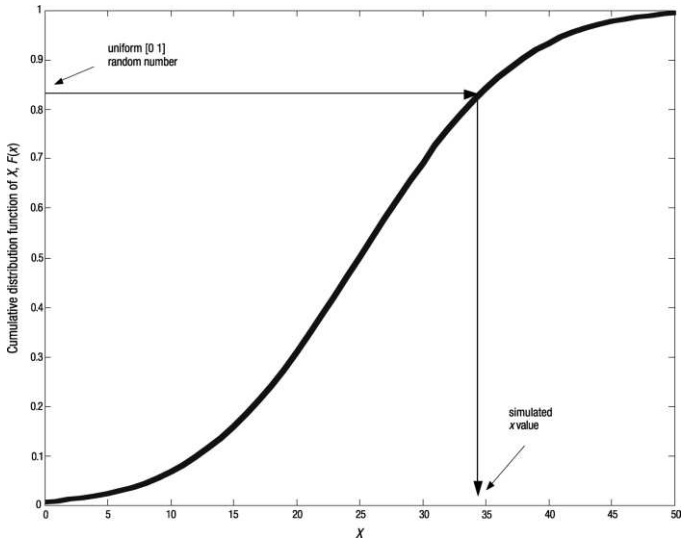


Fig. 15. Univariate Monte Carlo simulation using the CDF of the variable to be estimated (Avseth et al., 2005).

4.2.7. Running Monte Carlo simulations in Python

For running Monte Carlo simulations in Python, we need:

- The cumulative distribution functions (CDFs). These are described as NumPy arrays with elements or probabilities from 0 to 1.
- A random number generator from a uniform distribution: we used the *random* module of Python, and the random sample is drawn from the *random.uniform* function.

We created the *montecarlo* function, which works as follows. First, we define a list with four elements [0, 1, 2, 3]. Numbers 1, 2, and 3 are related to axis, off-axis and margin architectural positions respectively. Then, we create an empty list, that is going to store every value of the n iterations. We compare the architectural positions' numerical list [0, 1, 2, 3] with the cumulative sum of the architectural positions' probabilities [w, x, y, z] defining three options (axis, 1; off-axis, 2; and margin, 3), each associated with one of the three slope segments of the CDF. Later, we draw a random value from a uniform distribution with values from 0 to 1 using the *random.uniform* function. Finally, If the value falls between w and x is going to be assigned a value of 1 (i.e., axis). If the value falls between x and y is going to be assigned a

value of 2 (i.e., margin). If the value falls between y and z is going to be assigned a value of 3 (i.e., margin) (Fig. 16).

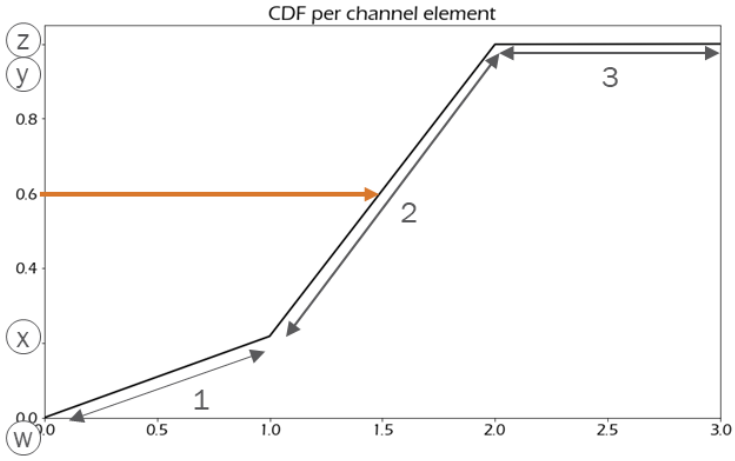


Fig. 16. Cumulative distribution function (CDF) of one of the channel element architectural positions within the measured section CACH1. When a value is drawn from a uniform distribution that goes from 0 to 1, one of the three possible architectural positions is going to be assigned. In this case, the randomly generated value of 0.6 (orange arrow), is going to be assigned an off-axis position (2). The three architectural positions are limited by changes in the slope of the line segments that describe the CDF.

This process is repeated n iterations, producing n simulated values or samples, and the results are stored in a list. Then, we can calculate the probability of occurrence of axis, off-axis and margin positions by counting the number of times that the random process yielded 1, 2, and 3 respectively and dividing that by the number of simulated samples or iterations.

4.3. Results

4.3.1. Probability density functions and cumulative distribution functions

The *pdfs_and_cdfs* function plots the PDFs and CDFs distribution of the architectural position probabilities per channel element within the measured section from the geological or stratigraphical base to the top and was used in the measured section CACH1 (Fig. 17).

4.3.2. Testing Monte Carlo simulation

The Monte Carlo simulation is performed for each channel element, using the corresponding architectural position probabilities. In the end, a histogram showing the distribution of the simulated samples is generated for each channel element, again from bottom to top. We can now compare the PDFs, CDFs and histograms (Fig. 18) to see if the estimated and original values have similar statistical distributions.

By plotting the probability density functions (PDFs) and the cumulative distribution functions (CDFs) in one column, and the histograms of the distribution of the simulated values from the Monte Carlo simulation in another column, we can see that the distribution of the results from the Monte Carlo simulation is similar to the original probability density functions (PDFs). It is important to plot the PDFs, CDFs and histograms from bottom to top, so we can also add the geological information provided by the measured section and see graphically the relationship between the architectural position probabilities and the channel elements stacked within the measured section CACH1 (Fig. 18).

4.3.3. Monte Carlo simulation of multiple equiprobable stacking patterns

We built several realizations of channel stacking patterns based on a random draw on the CDFs of the different channel elements constrained to the probabilities of classification for axis, off-axis and margin from the neural network. The results are multiple equiprobable realizations that are conditioned to probabilities at the wellbore. From them, the individual architectural positions of the channel elements were reclassified (i.e., the final architectural positions were gotten from the maximum number of occurrences from 10000 iterations over the CDFs for every channel element).

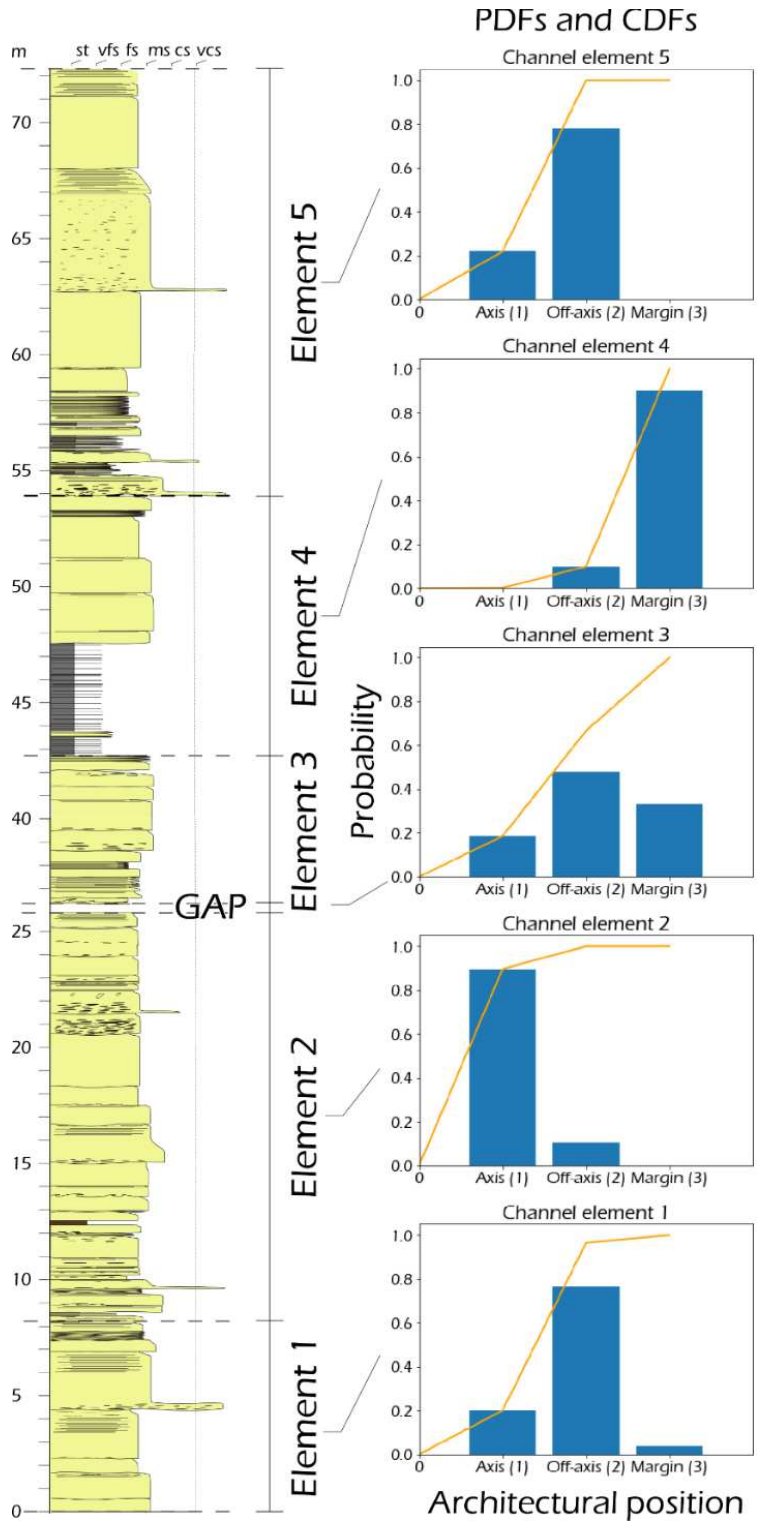


Fig. 17. From left to right, measured section CACH1, showing the actual layers observed in the field and the tops and bases of the five channel elements within the measured section. In analyzing the measured sections, we are assuming no gaps (a continuous record). In the second column, the PDFs, or bars of the architectural position probability of classification, or occurrence. Orange lines are the CDFs. It is important to generate the PDFs and CDFs from bottom to top so they can be easily linked to the geological observations.

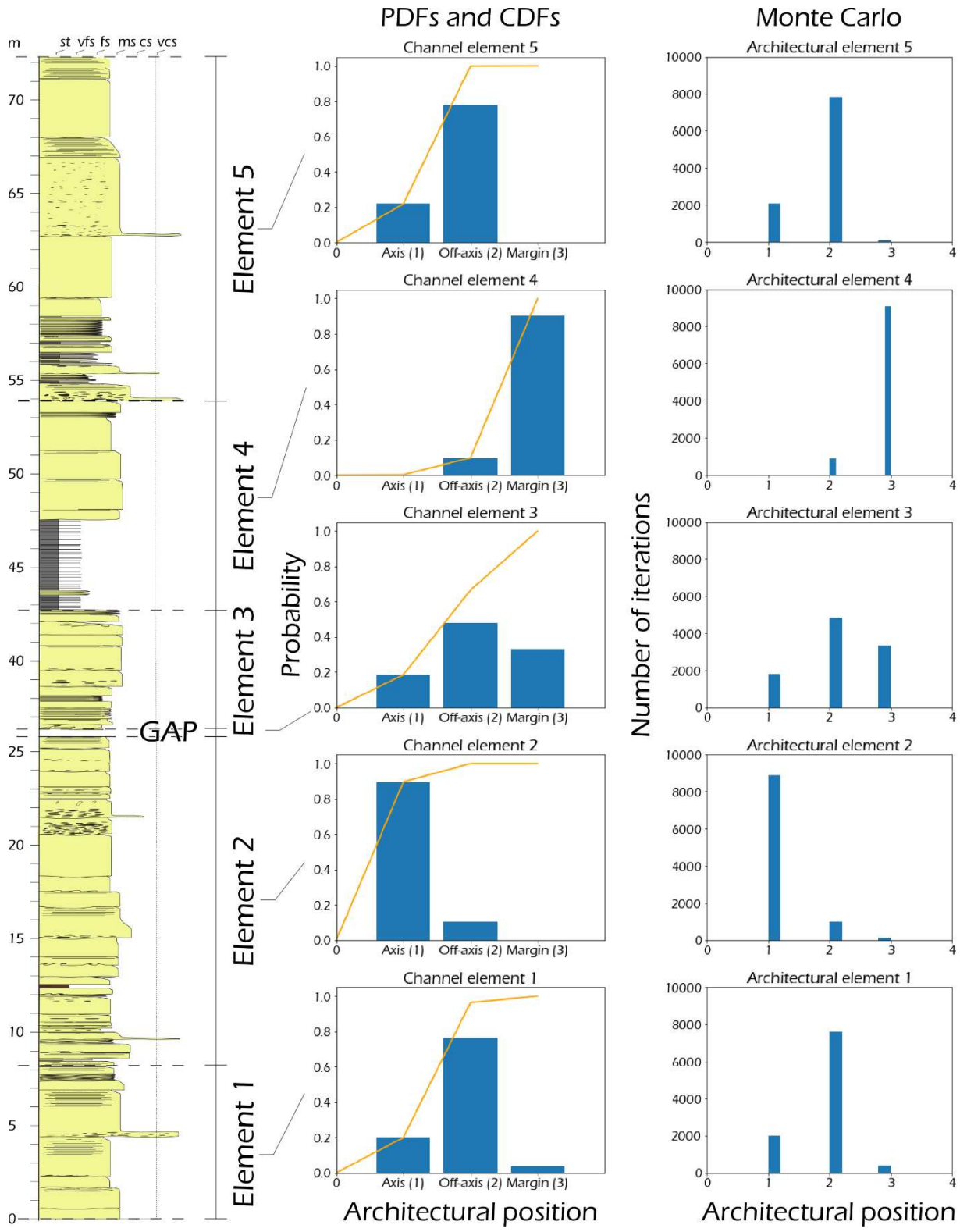


Fig. 18. From left to right, first column, measured section CACH1. In the second column, the PDFs, or bars of the architectural position probability of classification, or occurrence. Orange lines are the CDFs. In the third column, histograms show the distribution of the simulated samples. Notice that the distribution of those histograms is similar to the PDFs.

4.3.4. Channel stacking pattern templates

As mentioned above, parabolas have proved to be a convenient way for representing and modeling channel elements (Li and Caers, 2011; Morris et al., 2022; Rongier et al., 2017; Sylvester et al., 2011). From Eq. 1, we can model several parabolas at once relating the parabolas' vertex to the total offset from two components, the lateral offset (x) and vertical offset (y). Nevertheless, we would have infinite options for both the lateral offset (x) and vertical offset (y). Or at least, if we control the vertex on an x, y plane from the origin (i.e., x=0, y=0 coordinate), the lateral offset (x) would be a number between -200 m and 200 m to fit any architectural position within a measured section, and the vertical offset (y), a number between 0 m and the cumulative thickness of the specific measured section.

Since the channels of the Tres Pasos Fm. are low-sinuosity, high-symmetry channels, there are at least five architectural positions: the left margin, the left off-axis, the axis, the right off-axis and the right margin. From the channel dimensions used by Ruetten (2021), the left and right margins were assigned each 12.5% of the parabola, or 50 m on the x-axis. The left and right off-axis were assigned each 7.5% of the parabola, or 30 m in the x-axis. This is the narrowest architectural position, and as mentioned above, the transition from margin to axis occurs over the 30 m in the x-axis of the off-axis architectural position (Macauley and Hubbard, 2013). Finally, the axis was assigned 60% of the parabola, or 240 m on the x-axis. All those distances regarding the x-axis sum up 400 m, the width of the individual channel element template.

Five possible lateral offsets, directly related to the five possible architectural positions were modeled from the middle distances on the x-axis per architectural position to the y-axis. Then, the lateral offset (x) of the axis is 0 m, the lateral offsets for the left and right off-axis are -135 and 135 m respectively, and the ones for the left and right margins are -175 and 175 m respectively. The five possible lateral offsets/architectural positions are shown in Fig. 19.

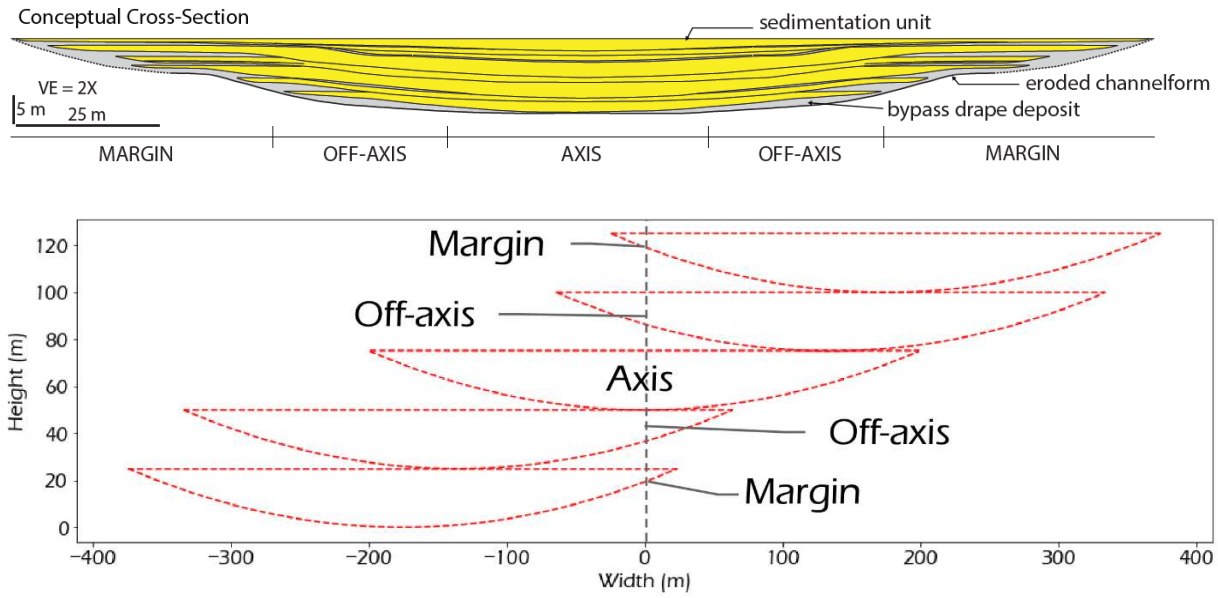


Fig. 19. At the top, the conceptual cross-section of channel element architectural positions from Macauley and Hubbard (2013); at the bottom, are templates of the five possible lateral offsets, which in turn are related to at least five architectural positions. These sketches were created with a hypothetical vertical offset (y) of 25 m between parabolas for visualization purposes.

Regarding the vertical offset (*y*), we can choose any number, as long as our geological thought remains valid for a particular situation. Setting up a vertical offset of 25 m means perfect preservation of the tops and bottoms of the channels, which is rare. Rather than that, usually, the tops of the channels are eroded as the accommodation space for the next channel is created, so vertical offsets of less than 25 m will make more sense. Other times, channels are stacked just with lateral offset (*x*), so no or little vertical offset would be required. Fig. 20 shows the different channel stacking patterns for some hypothetical vertical and lateral offsets. Notice that you have many theoretically infinite possibilities for channel stacking patterns, but not all of them have geological significance.

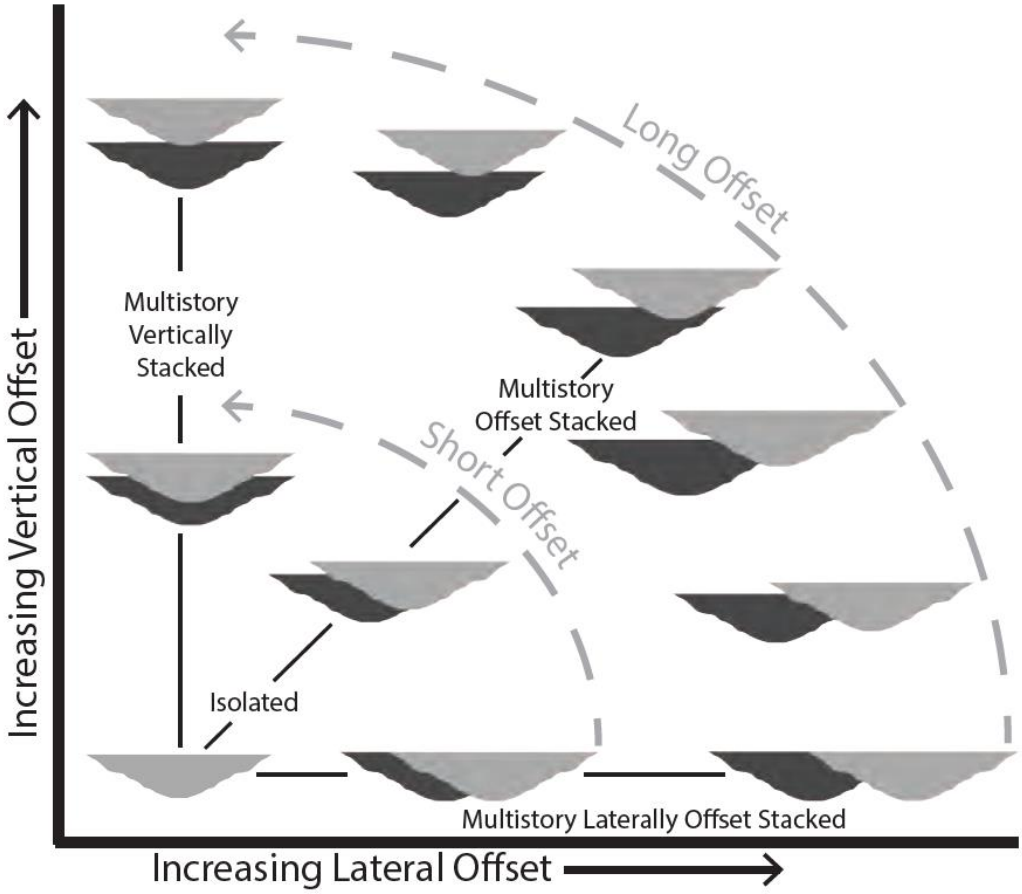


Fig. 20. The number of channel element stacking patterns is theoretically infinite but can be described by the vertical and lateral offsets. It requires geological thinking to match the information provided by the measured sections. From Hubbard et al. (2023).

4.3.5. Channel stacking pattern generation

For generating several realizations of channel stacking patterns, considering that there are two margins (left and right) and two off-axis (left and right), and understanding the visualization limitation inherent to measured sections, or well log data (i.e., allow us to see information as vertical profiles, or 1D), we define the function *stacking_patterns*. Within that function, it is required to define the final chain or list of numbers with the numerical code of the architectural positions from the Monte Carlo simulation, the number of random stacking patterns you want to visualize, and a constant vertical offset that makes sense with geological reasoning. Furthermore, it is possible to change the distances of lateral offsets by defining

a percentage from the axis to the final distance of a specific architectural position. In the end, a list of lateral and vertical offsets is generated.

For generating the list of lateral offsets, an empty list is first defined, then for each channel element in the measured section, a random choice between left and right (for margin and off-axis architectural positions) is selected using the *random.choice* function. Each result of the loop is attached using the *append* function. The list of vertical offsets is generated by creating an empty list, and then multiplying a constant vertical offset by the number of cumulative channel elements within the section from bottom to base. Each result of the loop is attached using the *append* function.

For plotting the randomly generated channel stacking patterns, we define a list of lateral and vertical offsets from the results of the *stacking_patterns* function, and another line of code takes these two lists and generates random channel stacking patterns with constant vertical offsets. Randomly generated channel stacking patterns for a constant 25 m vertical offset of the measured section CACH 1 are shown in Fig. 21.

Random channel stacking patterns with constant vertical offsets of 20, 15 and 10 m are shown in Fig. 22, Fig. 23 and Fig. 24 respectively, to illustrate the differences in total thickness if we decrease the vertical offset. Overall, also notice that some channel stacking patterns look more “harmonic”, in contrast with other channel stacking patterns that look more “disorganized”.

Running an experiment with 100 realizations would be robust enough to be able to depict the number of possible channel stacking patterns because the number of possible channel stacking patterns n for this measured section is 2 x 2 x 1 x 2 x 2 or = 16 when modeling individual channel elements first. Then, using the plotting code line for depicting the randomly generated channel stacking patterns, we got a figure with all the possible channel stacking patterns for the measured section from the list of unique possible lateral offsets and the constant vertical offsets defined in the *stacking_patterns* function results (Fig. 25).

Again, some possibilities look perfectly “harmonic” (Fig. 25, stacking patterns 8 and 9), whereas others look more “disorganized”.

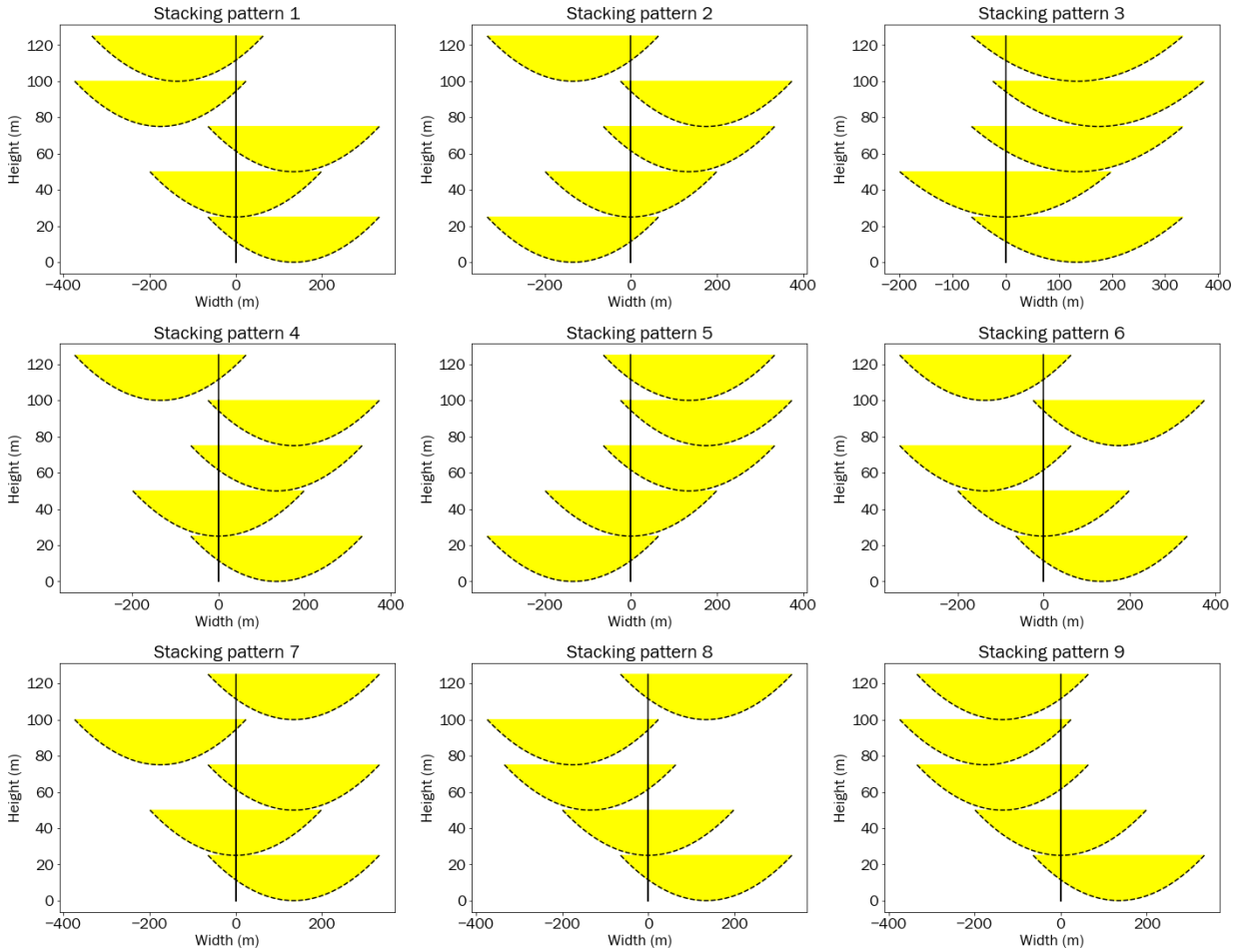


Fig. 21. Nine random realizations of channel stacking patterns for the measured section CACH1, with a constant vertical offset of 25 m. Notice that some channel stacking patterns look more "harmonic" (channel stacking patterns 5 and 9 in this figure), whereas in others the lateral offset between two channel elements exceeds half-width of the previous channel element. The black lines denote the trace of the measured section.

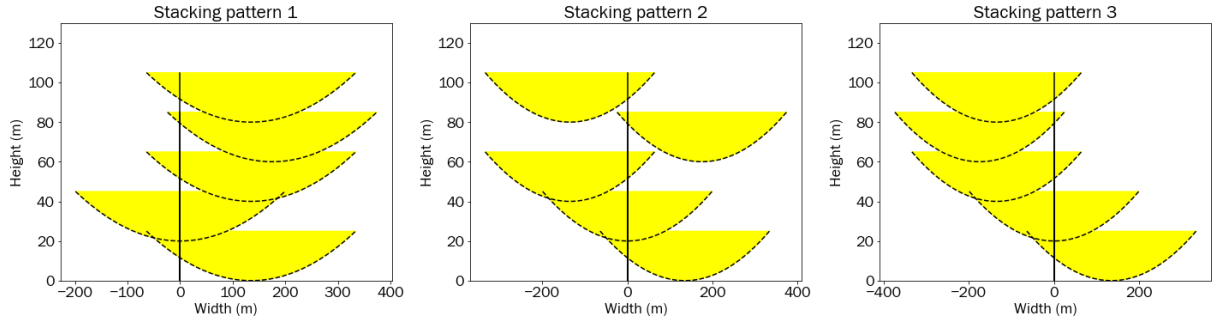


Fig. 22. Three random realizations of channel stacking patterns for the measured section CACH1, with a constant vertical offset of 20 m. Notice that some channel stacking pattern three looks more "harmonic", and also the total thickness decreases. The black lines denote the trace of the measured section.

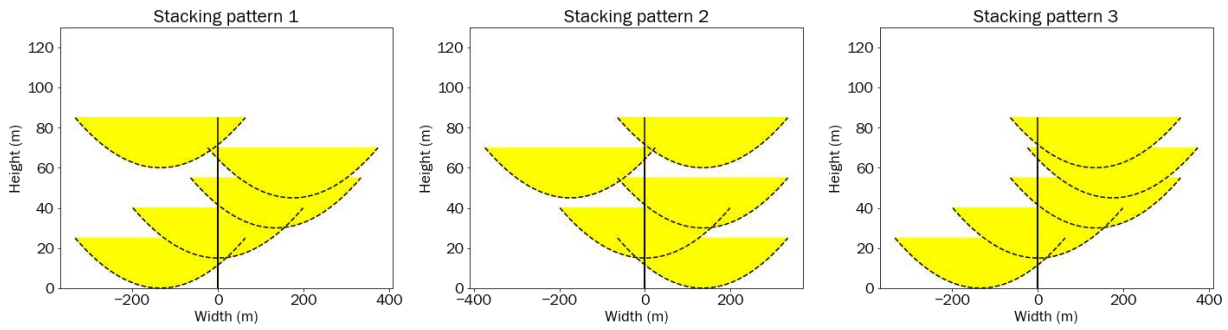


Fig. 23. Three random realizations of channel stacking patterns for the measured section CACH1, with a constant vertical offset of 15 m. Notice that some channel stacking pattern three looks more "harmonic", and the total thickness decreases. The black lines denote the trace of the measured section.

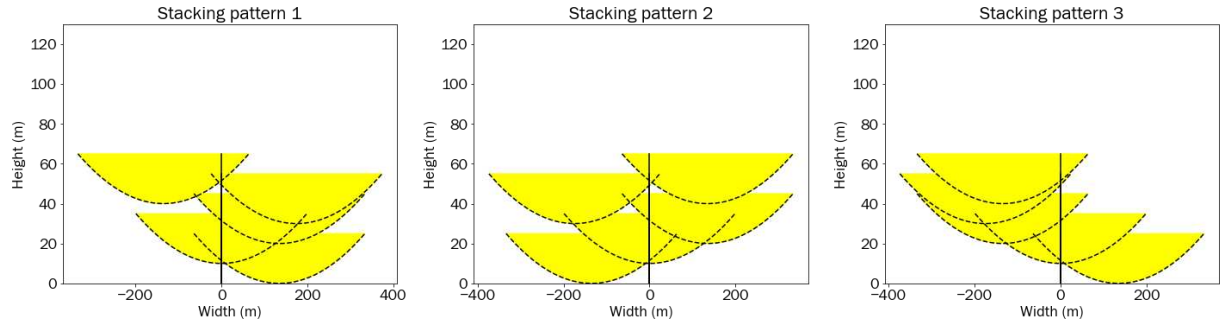


Fig. 24. Three random realizations of channel stacking patterns for the measured section CACH1, with a constant vertical offset of 10 m. Notice that some channel stacking pattern three looks more "harmonic", and the total thickness decreases. The black lines denote the trace of the measured section.

4.3.6. Channel stacking generation from Monte Carlo simulation

The channel stacking is modeled (rather than the individual channel elements and then the channel stacking) using the soft probabilities. From statistics, the total number of possible channel stacking patterns (n) with a channel template with five possible architectural positions (or five possible lateral offsets) and five stacked channel elements is n = 5 x 5 x 5 x 5 x 5 or 5^5 = 3125. Using the *var_levels* function, which runs several simulations from a list of iterations and plots the unique channel stacking pattern possibilities vs. the total number of iterations per simulation (Fig. 26) we can see that even 1000000 realizations of

random channel stacking given the soft probabilities cannot capture all possible channel stacking (3125 possibilities).

From the results of the simulation with 1000000 iterations, we sorted and filtered the 10 most likely channel stacking possibilities (Table 1). The probability of occurrence of those channel stacking patterns is about 1.4% out of 1000000 iterations. Then we plotted the results as channel stacking patterns without matching to thickness (Fig. 27).

Table 1. The 10 most likely channel stacking possibilities. Elem: channel element from base to the top, -2: left margin, -1: left off-axis, 0: axis, 1: right off-axis, 2: right margin.

	Elem1	**Elem2**	**Elem3**	**Elem4**	**Elem5**	**Count**
1	-1	0	1	2	1	**14688**
2	-1	0	-1	2	1	**14594**
3	1	0	1	-2	1	**14566**
4	-1	0	-1	-2	-1	**14521**
5	-1	0	1	-2	1	**14509**
6	-1	0	-1	-2	1	**14470**
7	1	0	1	2	-1	**14424**
8	-1	0	1	2	-1	**14355**
9	1	0	1	2	1	**14335**
10	1	0	-1	2	1	**14331**

Following a similar procedure, from the results of the simulation with 1000000 iterations, we sorted and filtered the 10 least likely channel stacking possibilities (Table 2). The probability of occurrence of those channel stacking patterns is 0.0001% out of 1000000 iterations. They are unlikely. Then we plotted the results as channel stacking patterns without matching to thickness (Fig. 28).

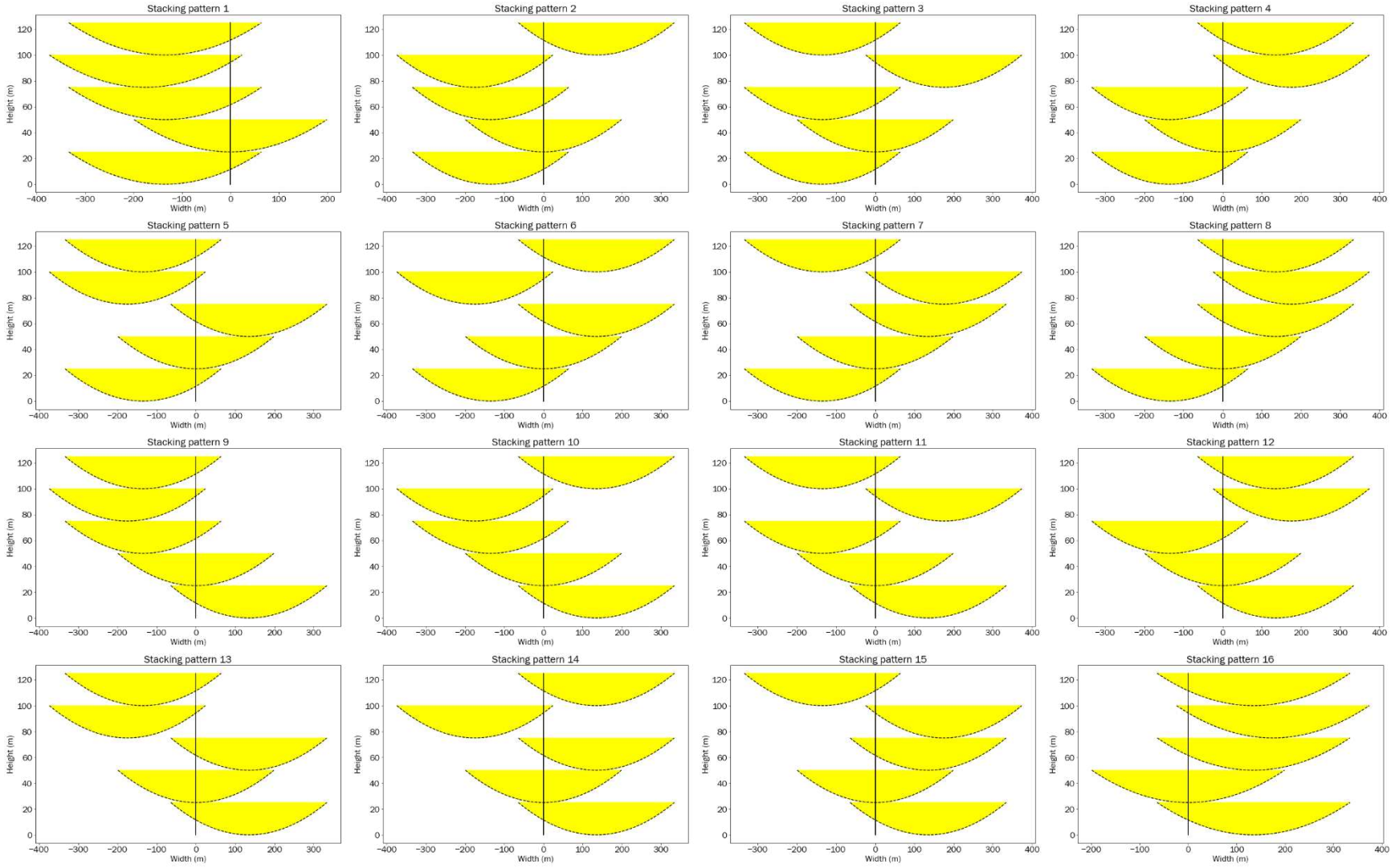


Fig. 25. Unique channel stacking pattern possibilities for the measured section CACH1. A constant vertical offset of 25 m between channel elements was used for visualization purposes. The channel stacking pattern possibilities are sorted from the one that has a more negative lateral offset (Stacking pattern 1) to the possibility that has a more positive lateral offset (Stacking pattern 16).

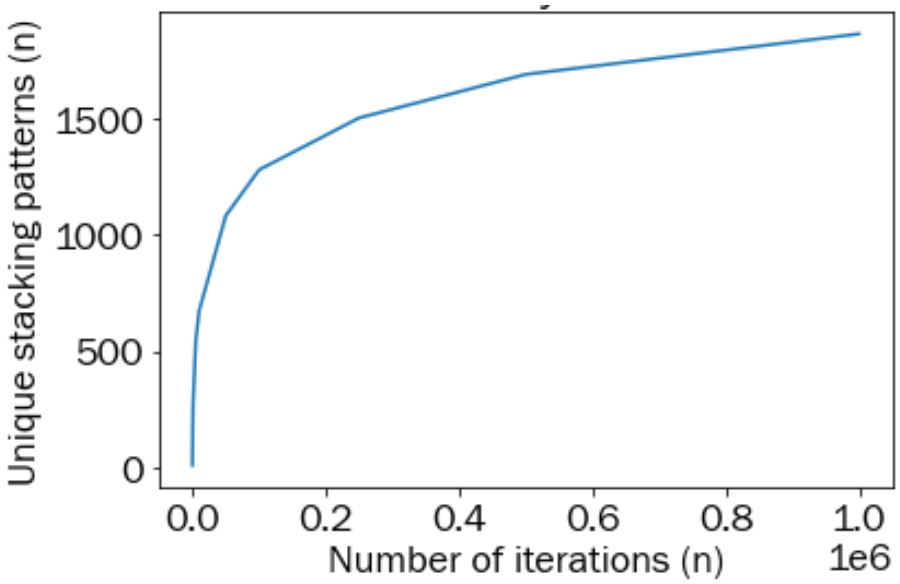


Fig. 26. Variability levels of several simulations of channel stacking. The curve was plotted from the results of performing 10, 100, 500, 1000, 5000, 10000, 50000, 100000, 250000, 500000 and 1000000 iterations per simulation. Notice that it is not possible to get the total number of possible channel stacking (3125 possibilities).

Table 2. The 10 least likely channel stacking possibilities. Elem: channel element from base to the top, -2: left margin, -1: left off-axis, 0: axis, 1: right off-axis, 2: right margin.

	Elem1	**Elem2**	**Elem3**	**Elem4**	**Elem5**	**Count**
1	0	-1	-2	-2	-2	**1**
2	0	-1	-2	-1	2	**1**
3	1	1	-1	1	-2	**1**
4	-2	2	-2	2	1	**1**
5	0	-1	-1	-1	2	**1**
6	2	-1	2	1	1	**1**
7	2	-1	2	2	-2	**1**
8	-2	2	-2	1	0	**1**
9	-2	2	-2	-2	0	**1**
10	2	-2	0	1	1	**1**

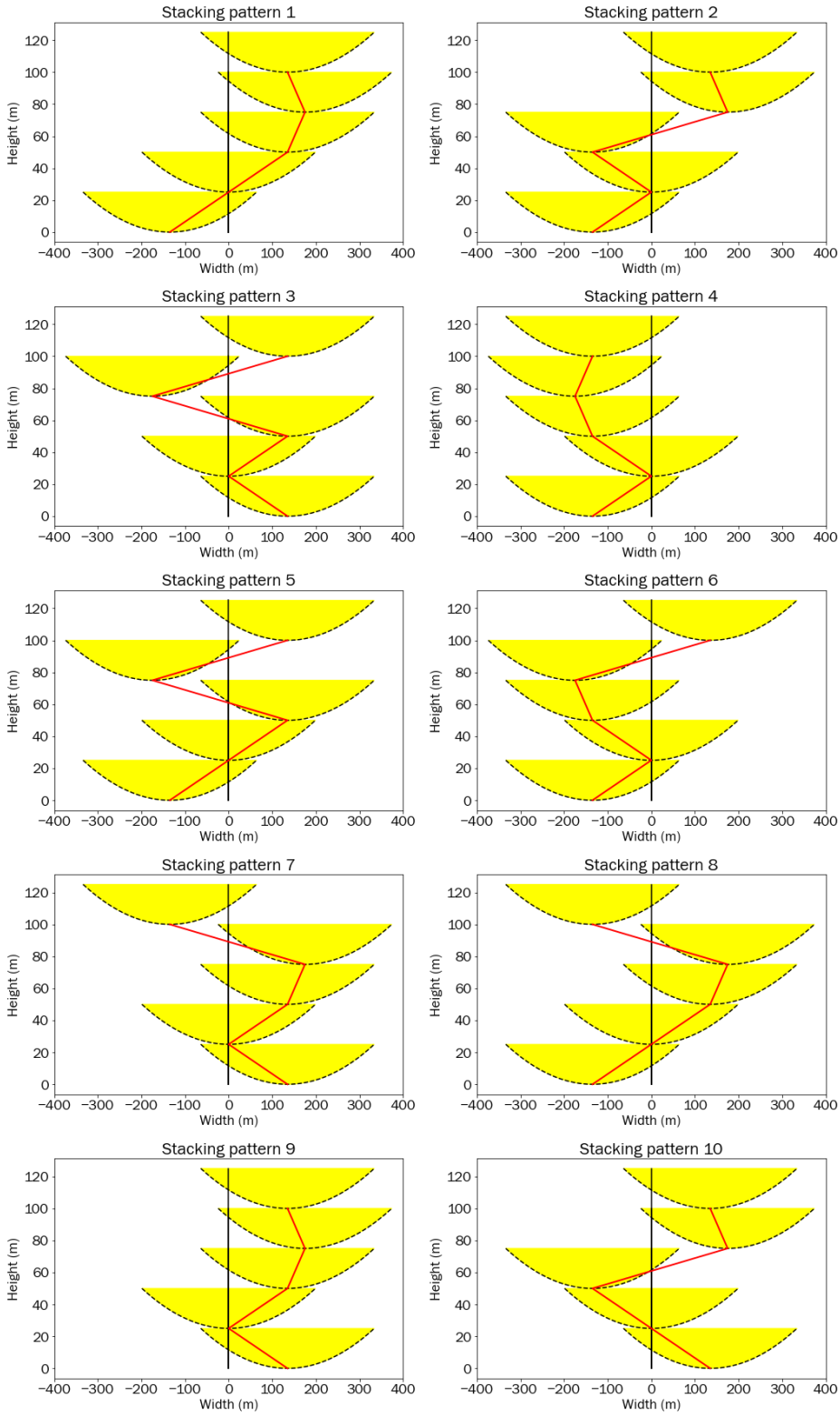


Fig. 27. The 10 most likely channel stacking pattern possibilities for the measured section CACH1, simulating the channel stacking directly from the soft probabilities. The channel stacking pattern possibilities are sorted from the one that has more occurrences out of 1000000 iterations to the possibility that has fewer occurrences. The black line is the trace of the measured section and the red line follows the parabola vertexes of the channel elements.

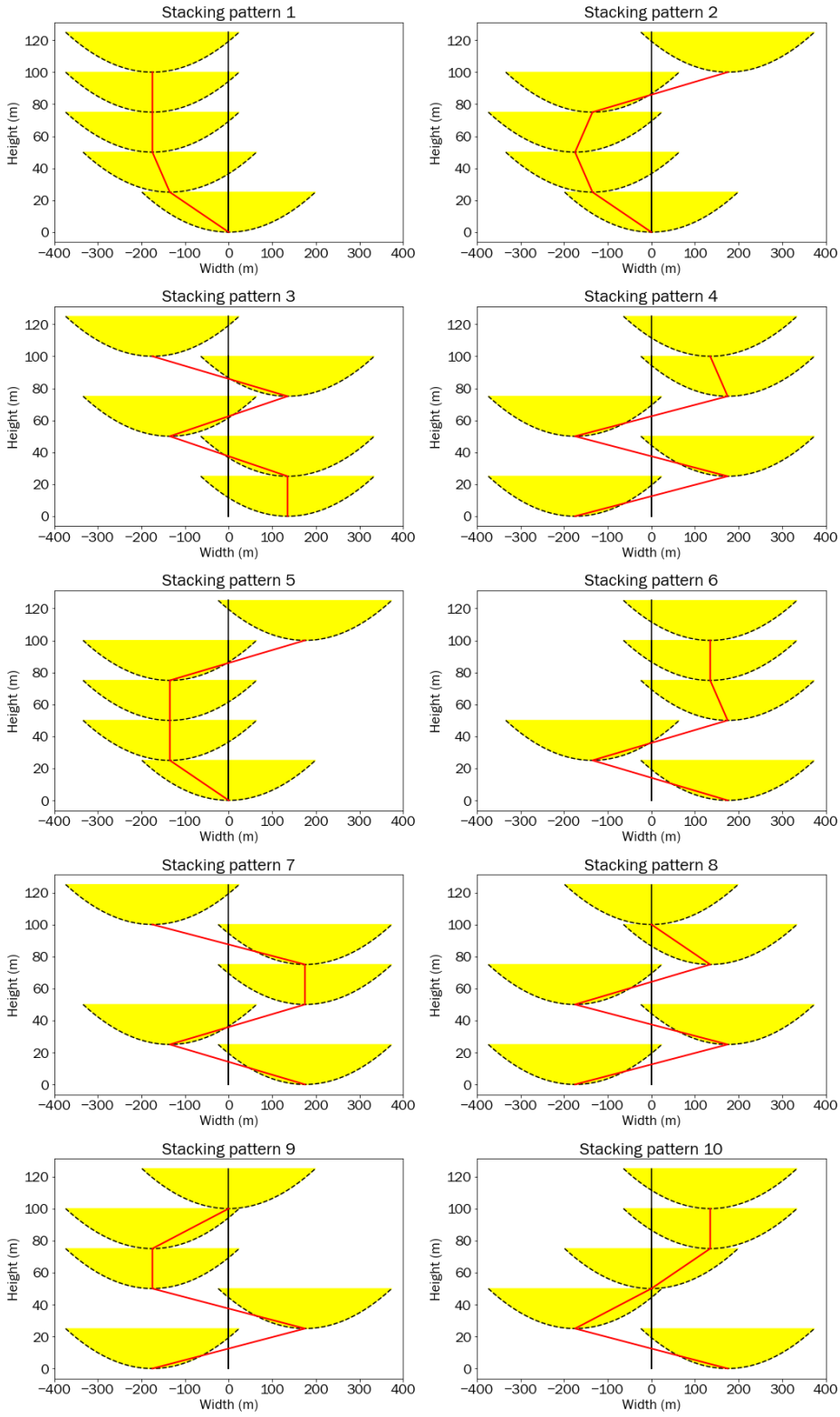


Fig. 28. The 10 least likely channel stacking pattern possibilities for the measured section CACH1, simulating the channel stacking directly from the soft probabilities. All these channel stacking pattern possibilities had just one occurrence out of 1000000 iterations. The black line is the trace of the measured section and the red line follows the parabola vertexes of the channel elements.

CHAPTER 5. MODELING CHANNEL STACKING PATTERNS WITH MARKOV TRANSITION PROBABILITIES

5.1. Motivation

Markov transition probabilities have been noted in many geological situations, including stratigraphic sequences of rocks and sedimentary processes. A Markov chain is a probabilistic model showing a special type of dependence, given the present condition, the future prediction does depend on the past. Previous studies have shown a strong dependence on the deposition of a given channel element by the composition and location of previous ones in organized channel systems (Macauley and Hubbard, 2013; McHargue et al., 2011a; Sylvester et al., 2011). Given that, the stacking of deep-water channels can be represented as a Markov chain, and channel stacking patterns can be predicted by using a vertical transition probability matrix directly obtained from measured sections.

With this approach, the prediction of the channel stacking pattern is given by the deposition of the first channel element, or in this case, a random seed, rather than being controlled by the soft probabilities from the neural network. This allows us to contrast the results from the conditional modeling with soft probabilities, test if the vertical transition probability matrix could be used to create organized vs. disorganized channel stacking patterns and see how the results would be if we used that matrix alone to create channel stacking patterns.

5.2. Methods

5.2.1. Database and coding style

For calculating the Markov transition probabilities, we used the information provided by the Chile Slope Systems consortium database (https://www.chileslopesystems.com/). After accessing the database, we downloaded a .xlsx copy of it. This database contains information of 785 channel elements, in contrast

with the machine learning-derived probabilities of classification of Vento (2020), which contains information of 154 channel elements from 64 measured sections.

We also performed a quality control of the database. From the original database, about 60% of architectural positions of the channel elements were 'NoneDefined', meaning that they had not been classified for either a geologist or a machine learning-related technique. There were full measured sections in which the 'NoneDefined' category existed. Nevertheless, there were 'NoneDefined' architectural positions within a measured section with some information. By cross-validation with the measured sections, sometimes these 'NoneDefined' intervals were related to mass transport deposits (MTDs, not considered in this research) but other times they do not.

We removed then all the measured sections with only 'NoneDefined' architectural position classes and the 'NoneDefined' classes of measured sections that still had some architectural positions classified, to focus on calculating the transition probabilities for axis, off-axis and margin architectural positions. In the end, we worked with the architectural position classifications of 157 channel elements from 64 measured sections, just a little bit more channel elements than the ones provided by Vento (2020).

Again, by removing all 'NoneDefined' classes, we are assuming a measured section without gaps, with a complete record and without stratigraphic vertical complexity (no mass transport deposits – MTDs occurrence). Then, we proceeded to upload the .xlsx file as a pandas.DataFrame in Python using JupyterLab. Code to generate channel stacking patterns with the information of a vertical transition probability matrix and a seed was created. We used a random choice between axis, off-axis and margin as the seed for the simulations. The code also allows to choose between one of them if required. We modeled five stacked channel elements. These results are unmatched to thickness. Finally, we created a function that calculates the total distance between the parabola vertexes per channel stacking pattern (*distances* function) as an indicator to compare channel stacking patterns with low vs. high lateral offset. The script was written in a functional programming style.

5.2.2. Markov transition probabilities

Markov transition probabilities have been noted in many geological situations, including stratigraphic sequences of rocks and sedimentary processes. Stratigraphy (i.e., the branch of geology that studies layered rocks, and in general, the distribution in time and space of rocks) can be represented as a Markov chain.

Let's consider layers of rocks at discrete points along a vertical profile. The points are numbered from bottom to top, and the use of the Markov transition probabilities assumes that the lithology of a younger rock depends on the lithology of the immediately older rock. Sometimes the same lithology is observed at successive layers of rocks. Then, the transition matrix that gives the probability of going from one lithology to another has nonzero probabilities on the diagonals (Krumbein and Dacey, 1969).

A Markov chain is a probabilistic model showing a special type of dependence, given the present condition, the future prediction does depend on the past. In 1D situations, as the geological record is shown in measured sections and well logs, a Markov chain is represented by a vertical transition probability matrix. Transition probabilities correspond to the frequencies of transitions from a certain layer of rock to another layer of rock and are arranged in a square matrix form (Fig. 29). Those probabilities are called single-step transitions (Elfeki and Dekking, 2001).

$$\boldsymbol{p} = \begin{bmatrix} p_{11} & p_{12} & \cdot & \cdot & p_{1n} \\ p_{21} & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & p_{lk} & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ p_{n1} & \cdot & \cdot & \cdot & p_{nn} \end{bmatrix}$$

Fig. 29. Example of a square matrix with transition probabilities (Elfeki and Dekking, 2001).

Given that measured sections and well logs are represented as vertical profiles, we can estimate the vertical transition probability matrix directly from them. First, the vertical profile (either measured sections or well logs) is divided into channel elements, each of them with its corresponding architectural position. Then a vertical transition count matrix is obtained by classic statistics, by counting the number of possible pairs. Dividing the number of possible pairs by the total number of pairs or transitions, in this case in the vertical direction, the vertical transition probability matrix is finally obtained (Cao et al., 2021; Qi et al., 2016).

5.2.3. Vertical transition count matrix

For getting the vertical transition count matrix, after uploading the .xlsx file CSS_DB_v0.2_ElementData_filtered.xlsx as a pandas.DataFrame in Python using JupyterLab, we converted this DataFrame into a NumPy array with the *values* function. Then, we got a list of the measured sections using the *unique* function.

We need to perform the counting of pairs per measured section, not for the whole database. If we did it for the whole database, extra pairs would appear because of counting the information between measured sections.

An empty list is generated, and for each measured section within the database, we create a list with the numerical code of the architectural position (i.e., axis = 1, off-axis = 2 and margin = 3). Then we define transition possibilities or pairs per measured section by using the *trans_prob_pairs* function. This function generates two NumPy arrays per measured section, one with all the first elements of the list but the last, and another with all the last elements of the list but the first. Then, these two arrays are stacked using the *stack* function. Finally, this function returns the list of transition possibilities or pairs per measured section.

With the final counting of transition possibilities or pairs, we concatenated all the lists from all the measured sections to then unpack them. In the end, a NumPy array with the pairs is obtained. The *unique* function allowed to get the list of possible transition pairs. Then, we performed the counting of the transition

pairs by generating an empty list and running a loop for counting the number of elements in the final transition pairs that match the list of unique pairs. A list of the counting is gotten. 93 transition pairs of 9 possible transition pairs were counted out of 157 channel elements from 64 measured sections.

For visualization purposes, we converted the list to a pandas.DataFrame and processed the results into a separate .xlsx file. The results take all the available data, without evaluating any statistical stationarity (e.g., changes in stacking pattern throughout the study area).

5.2.4. Vertical transition probability matrix

The previous results provided us with the vertical transition probability matrix, by dividing the number of pairs counted by the number of total pairs (93 pairs). We also converted the final NumPy array to a pandas.DataFrame and processed the information in a .xlsx file for formatting purposes.

5.3. Results

5.3.1. Vertical transition count matrix

The final vertical transition count matrix is shown in Table 3. Hot (red) colors indicate more pairs counted. Cool (green) colors indicate fewer pairs counted.

Table 3. Vertical transition count matrix for the Chile Slope Systems consortium database.

	Axis	**Off-axis**	**Margin**
Axis	9	18	5
Off-axis	15	17	7
Margin	8	7	7

5.3.2. Vertical transition probability matrix

The final vertical transition probability matrix is shown in Table 4. Hot (red) colors indicate higher probabilities. Cool (green) colors indicate lower probabilities. As expected, the sum of all the possible probabilities (all cells in Table 4) is equal to 1.

Table 4. Vertical transition probability matrix for the Chile Slope Systems consortium database.

	Axis	**Off-axis**	**Margin**
Axis	0.1	0.19	0.05
Off-axis	0.16	0.18	0.08
Margin	0.09	0.08	0.08

5.3.3. Normalized vertical transition probability matrix

We also generated a normalized vertical transition probability matrix (Table 5). The values per row were normalized to 1, so it can be seen specifically, for example, what the probabilities of transitioning vertically from axis to axis, off-axis and margin are (Table 5, first row).

Table 5. Normalized vertical transition probability matrix for the Chile Slope Systems consortium database.

	Axis	**Off-axis**	**Margin**
Axis	0.29	0.56	0.15
Off-axis	0.38	0.43	0.19
Margin	0.36	0.32	0.32

From Table 5, it is 56% more likely to transition vertically from axis to off-axis architectural position. Transitioning to axis (29%) or margin (15%) is also possible but less likely. From an off-axis architectural position, it is more likely to transition vertically to off-axis (43%) or axis (38%). It is less likely to transition vertically into margin (19%) but still possible. Finally, transitioning from a margin to either an axis, off-axis and margin position (36%, 32% and 32% respectively) look equiprobable.

5.3.4. Sensitivity analysis with Markov transitional probabilities

We tested the code using several hypothetical vertical transition probability matrices to test if the vertical probability transition matrix could be used to generate organized vs. disorganized channel stacking patterns and evaluate what the most and least likely channel stacking patterns would be.

For the first hypothetical vertical transition probability matrix (Table 6), the probability of transitioning from axis to axis, from off-axis to off-axis and from margin to margin is equal to 1, and the probabilities of transitioning to the rest of possibilities is zero.

Table 6. First hypothetical vertical transition probability matrix for generating channel stacking patterns from a seed.

	Axis	**Off-axis**	**Margin**
Axis	1.00	0.00	0.00
Off-axis	0.00	1.00	0.00
Margin	0.00	0.00	1.00

Using that hypothetical vertical transition probability matrix for generating channel stacking patterns from a random seed, and using the *var_levels* function, which runs several simulations from a list of iterations and plots the unique channel stacking pattern possibilities vs. the total number of iterations per simulation, we can see that after 500 realizations of random channel stacking given the hypothetical vertical transition probability matrix we get 65 possibilities (Fig. 30).

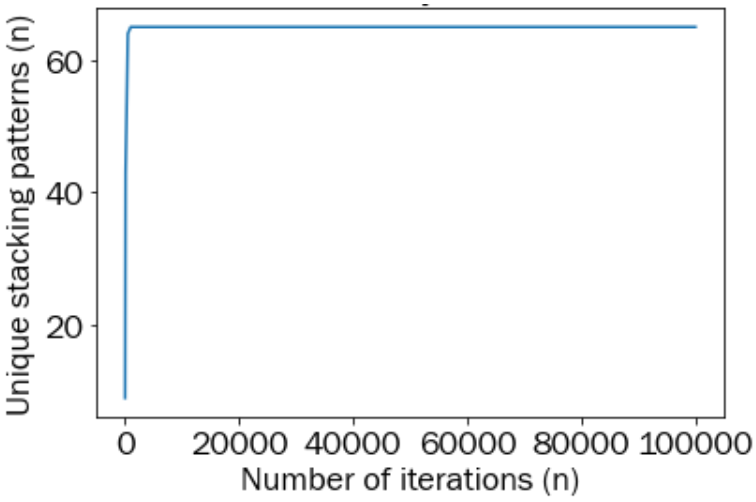


Fig. 30. Variability levels of simulations of channel stacking for the first hypothetical vertical transition probability matrix. The maximum number of iterations is reached after 500 realizations. The curve was plotted from the results of performing 10, 100, 500, 1000, 5000, 10000, 50000 and 100000 iterations per simulation.

The possible channel stacking patterns for this matrix are: (i) for the first channel element, there are 5 possibilities and (ii) given the seed, the channel stacking is going to get stuck within the same architectural position. If axis, the subsequent channels are going to be axis (1 possibility). If off-axis or margin, there are two possibilities associated with the left and right architectural positions (64 possibilities).

The 10 most likely channel stacking pattern possibilities using this matrix are shown in Fig. 31. The vertically aligned axis channel elements are by far the most likely because there is no uncertainty associated with the random selection of left or right architectural positions. 33423 occurrences were counted out of 100000 iterations. Next, random combinations between left and right off-axis and margins are the more likely channel stacking pattern. Vertically, the architectural position remains the same, but laterally, the random selection between left and right architectural positions creates random patterns. Those channel stacking patterns are about 1% likely out of 100000 iterations. The total distance, and therefore, the lateral offset between channel elements is variable (from 100 for the vertically stacked axis to 1077.67 for stacked off-axis and margin with high lateral offset, Fig. 31).

The 10 least likely channel stacking pattern possibilities using this matrix are shown in Fig. 32 Nevertheless, all realizations using this vertical transition probability matrix can be considered equiprobable, as expected. The 10 least likely channel stacking pattern possibilities also have a 1% probability of occurrence. The total distance is variable (from 346.15 to 1084.60 m, greater dispersion, Fig. 32).

In the second hypothetical vertical transition probability matrix (Table 7) the probability of transitioning from axis, off-axis and margin to off-axis is 1, and the rest of probabilities is zero. We wanted to illustrate a case in which all the architectural position transition towards the intermediate architectural position, or what could happen if the probabilities of transitioning to a fixed architectural position were the highest.

Using that hypothetical vertical transition probability matrix for generating channel stacking patterns from a random seed, and using the *var_levels* function, we can see that after 100 realizations of random channel stacking given the hypothetical vertical transition probability matrix, we get 80 possibilities (Fig. 33).

Table 7. Second hypothetical vertical transition probability matrix for generating channel stacking patterns from a seed.

	Axis	**Off-axis**	**Margin**
Axis	0.00	1.00	0.00
Off-axis	0.00	1.00	0.00
Margin	0.00	1.00	0.00

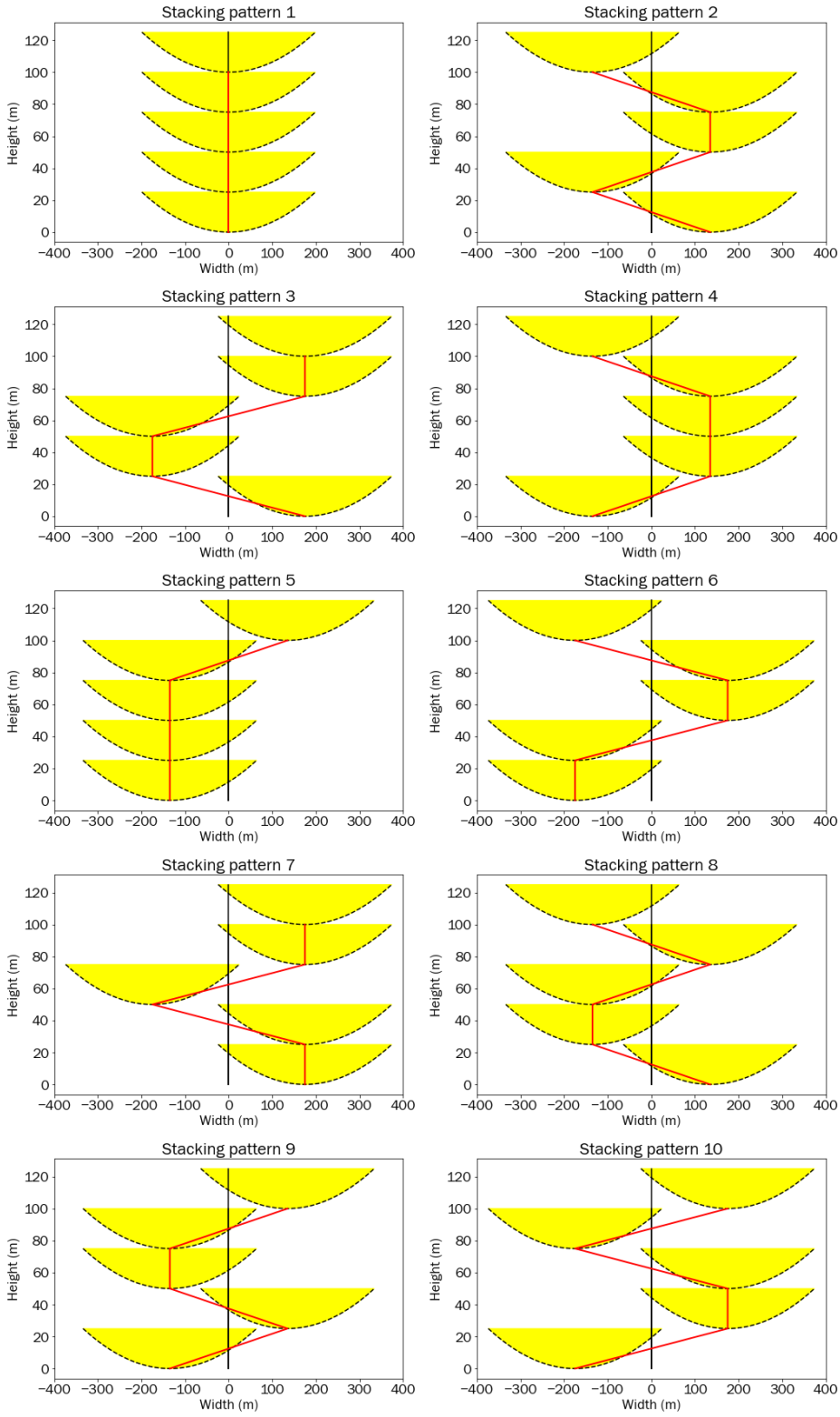


Fig. 31. The 10 most likely channel stacking pattern possibilities for five stacked channel elements using the first hypothetical vertical transition matrix. The channel stacking pattern possibilities are sorted from the one that has more occurrences out of 100000 iterations to the possibility that has fewer occurrences. The black line is the trace of the measured section and the red line follows the parabola vertexes of the channel elements

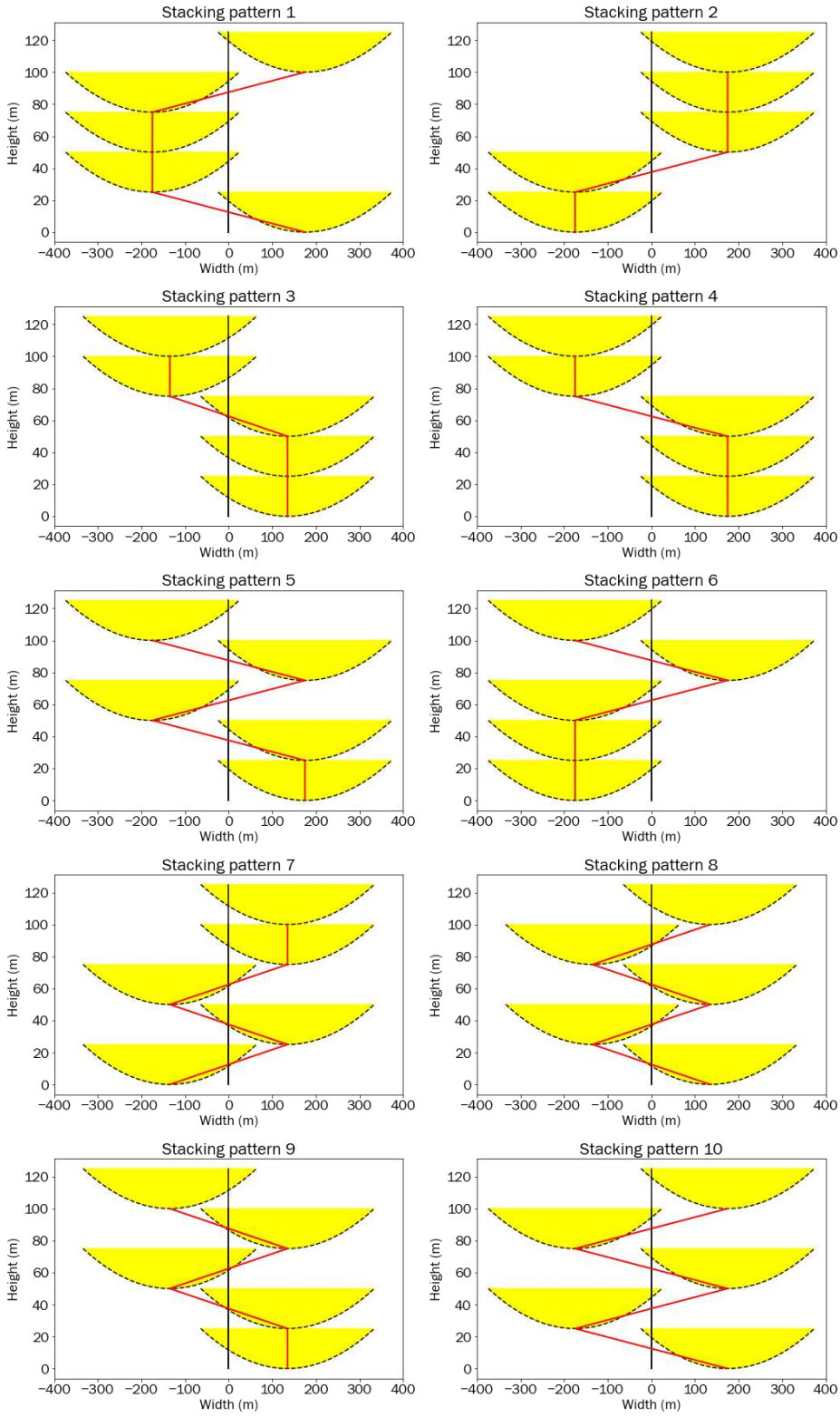


Fig. 32. The 10 least likely channel stacking pattern possibilities for five stacked channel elements using the first hypothetical vertical transition matrix. The channel stacking pattern possibilities are sorted from the one that has more occurrences out of 100000 iterations to the possibility that has fewer occurrences. The black line is the trace of the measured section and the red line follows the parabola vertexes of the channel elements.

The possible channel stacking patterns for this matrix are: (i) for the first channel element, there are 5 possibilities and (ii) given the seed, all subsequent channel elements are going to be either left off-axis or right off-axis (5 x 2^4 = 80 possible channel stacking patterns).

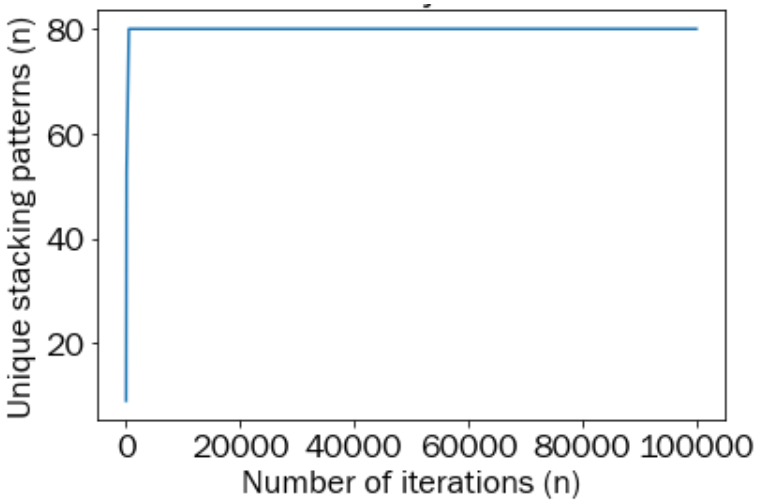


Fig. 33. Variability levels of simulations of channel stacking for the second hypothetical vertical transition probability matrix. The maximum number of iterations is reached after 100 realizations. The curve was plotted from the results of performing 10, 100, 500, 1000, 5000, 10000, 50000 and 100000 iterations per simulation.

The 10 most likely channel stacking pattern possibilities using this matrix are shown in Fig. 34. The channel stacking patterns starting with the axis architectural position are the most likely because there is not a random selection such as left or right axis (2% likely). Subsequent channel elements are assigned to the left or right off-axis. Those channel stacking patterns are about 1% likely out of 100000 iterations. The total distances vary between 212.30 and 950.75 m, but most channel stacking patterns have 704.60 m.

The 10 least likely channel stacking pattern possibilities using this matrix are shown in Fig. 35. All realizations using this vertical transition probability matrix can be considered equiprobable, as expected. The 10 least likely channel stacking pattern possibilities also have a 1% probability of occurrence. Using

this matrix, was introduced more randomness between the seed and the subsequently predictable channel elements. The total distance shows greater dispersion (from 122.17 to 1084.60 m).

In the third hypothetical vertical transition probability matrix (Table 8), the probabilities of transitioning from axis to margin, off-axis to off-axis and margin to axis are 1, and the rest of the probabilities are 0. We wanted to see if we could generate channel stacking patterns with larger lateral offset by assigning the axis to margin and margin to axis probability as 1.

Table 8. Third hypothetical vertical transition probability matrix for generating channel stacking patterns from a seed.

	Axis	**Off-axis**	**Margin**
Axis	0.00	0.00	1.00
Off-axis	0.00	1.00	0.00
Margin	1.00	0.00	0.00

Using that hypothetical vertical transition probability matrix for generating channel stacking patterns from a random seed, and using the *var_levels* function, we can see that after 500 realizations of random channel stacking given the hypothetical vertical transition probability matrix we get 44 possibilities (Fig. 36). The seed has five possibilities. If off-axis, the channel stacking pattern will swing between left and right off-axis (2 x 2 x 2 x 2 x 2 = 2^5 = 32 possibilities).

If axis, the stacking pattern is going to swing between axis in the odd channel elements and left and right margins in the even channel elements (1 x 2 x 1 x 2 x 1 = 4 possibilities). If margin, the stacking pattern is going to swing between left and right margin in the odd elements and axis in the even elements (2 x 1 x 2 x 1 x 2 = 8 possibilities). Therefore, 32 + 4 + 8 = 44 possible channel stacking patterns.

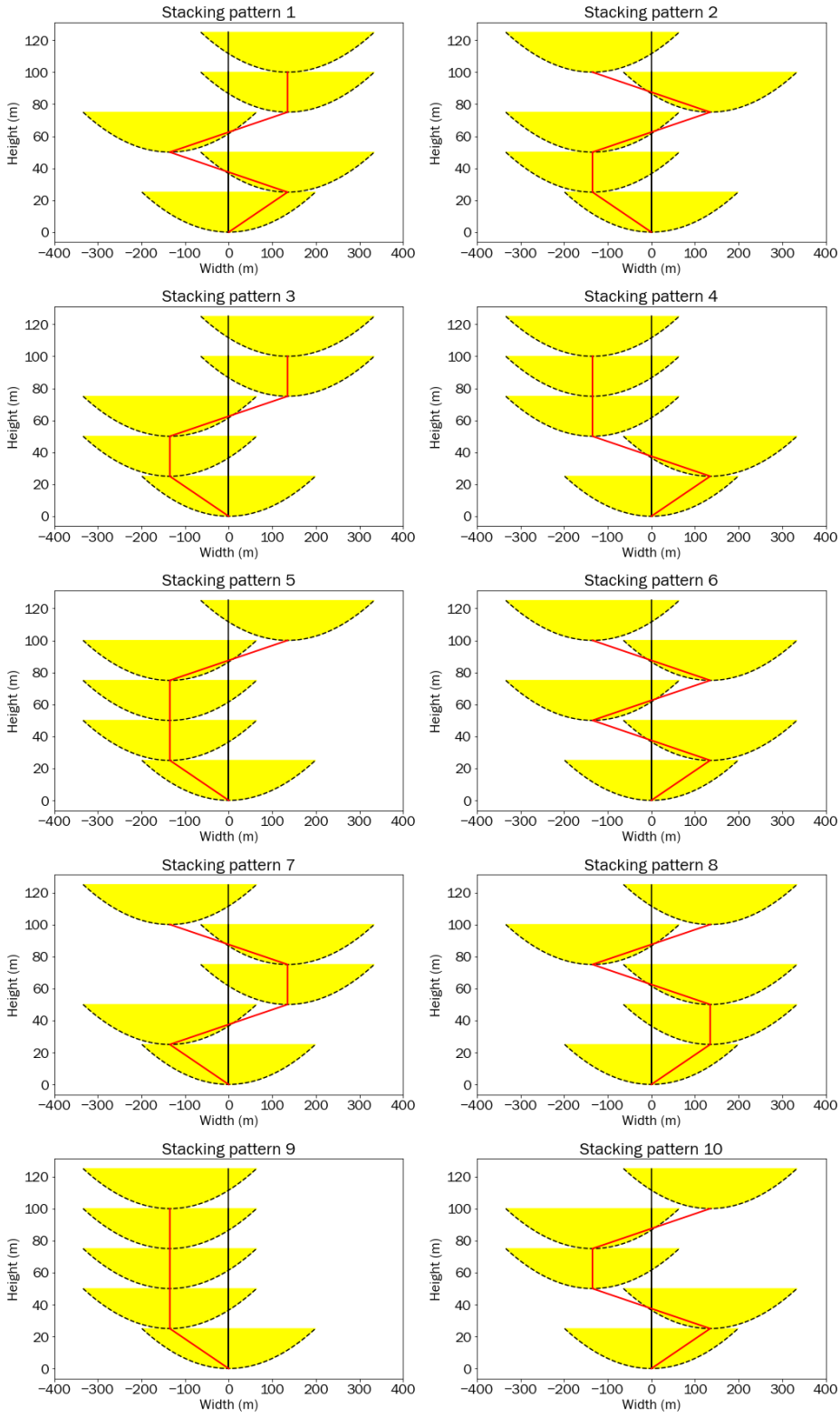


Fig. 34. The 10 most likely channel stacking pattern possibilities for five stacked channel elements using the second hypothetical vertical transition matrix. The channel stacking pattern possibilities are sorted from the one that has more occurrences out of 100000 iterations to the possibility that has fewer occurrences. The black line is the trace of the measured section and the red line follows the parabola vertexes of the channel elements.

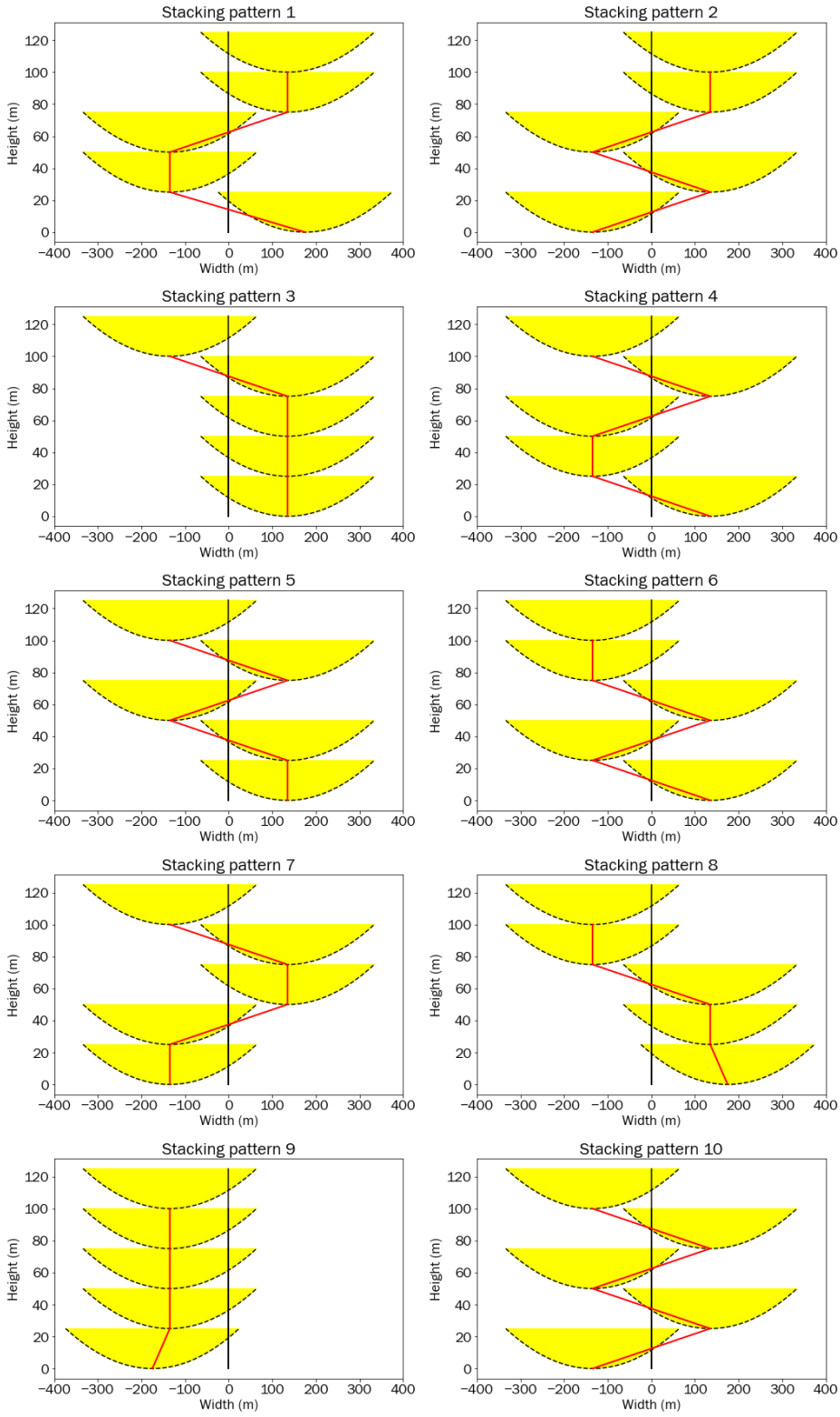


Fig. 35. The 10 least likely channel stacking pattern possibilities for five stacked channel elements using the second hypothetical vertical transition matrix. The channel stacking pattern possibilities are sorted from the one that has more occurrences out of 100000 iterations to the possibility that has fewer occurrences. The black line is the trace of the measured section and the red line follows the parabola vertexes of the channel elements.

The 10 most likely channel stacking pattern possibilities using this matrix are shown in Fig. 37. The channel stacking patterns involving axis to margin and margin to axis transitions are more likely. Their probability of occurrence ranges from 8% when axis is the seed to 4% when margin is the seed, as expected given the left or right margin random choice. The total distance of those possibilities is 702.12 m.

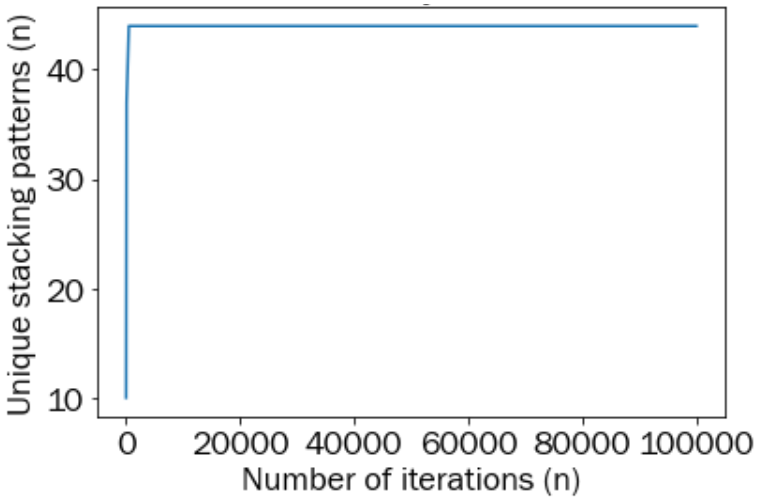


Fig. 36. Variability levels of simulations of channel stacking for the third hypothetical vertical transition probability matrix. The maximum number of iterations is reached after 500 realizations. The curve was plotted from the results of performing 10, 100, 500, 1000, 5000, 10000, 50000 and 100000 iterations per simulation.

The 10 least likely channel stacking pattern possibilities using this matrix are shown in Fig. 38. All combinations involving off-axis are the less likely, and their probability of occurrence is about 1%. The total distance in those stacking patterns is variable (from approx. 100 to 1000 m).

5.3.5. Channel stacking pattern construction from the outcrop statistics and a seed

Following robust testing of the channel simulation with transition probabilities, the outcrop-derived probabilities can be confidently used to generate simulations. The hypothesis is that the resulting simulations will more closely match stacking patterns from the outcrop. A caveat is, however, that the transition probabilities are derived from the full dataset, and an analysis of statistical stationarity is

considered. Channel stacking patterns vary throughout the study area. The resulting simulation does not capture any end-member stacking patterns and will not reproduce stacking patterns at the outcrop. Rather, the goal here is to demonstrate the use of outcrop-derived data in a near-wellbore simulation.

We used the real vertical transition probability matrix (Table 4) from the Chile Slope Systems consortium database for generating the most likely realizations of channel stacking patterns and quantifying their occurrence. We used a random choice between axis, off-axis and margin architectural positions as the seed, or first channel element architectural position and modeled five stacked channel elements.

Using the outcrop-derived vertical transition probability matrix for generating channel stacking patterns and the *var_levels* function, we can see that the total number of possible channel stacking patterns is reached after performing more than 50000 realizations. After that, the number of possibilities remains 3125, no matter the number of iterations (Fig. 39).

The 10 most likely channel stacking pattern possibilities using this matrix are shown in Fig. 40. Generating channel stacking patterns with the information from the real Markov vertical transition probability matrix, the most likely channel stacking patterns are the ones that involved intercalations between axis and off-axis architectural positions. The other ones more likely, are the ones in which there are amalgamated axis, with an occasional intercalation of off-axis. Those channel stacking patterns are around 4% likely and reflect the data from the real vertical transition count matrix (Table 3). The majority of transitional pairs counted are: from axis to off-axis (18 pairs), from off-axis to off-axis (17 pairs) and from off-axis to axis (15 pairs) and their transition probabilities are the highest (19%, 18% and 16% respectively).

The 10 least likely channel stacking pattern possibilities using the real vertical transition probability matrix are shown in Fig. 41. Vertically aligned intercalations of off-axis and margin architectural position are less likely (around 0.03% probability of occurrence). This also concurs with the data from the real vertical transition count matrix (Table 3).

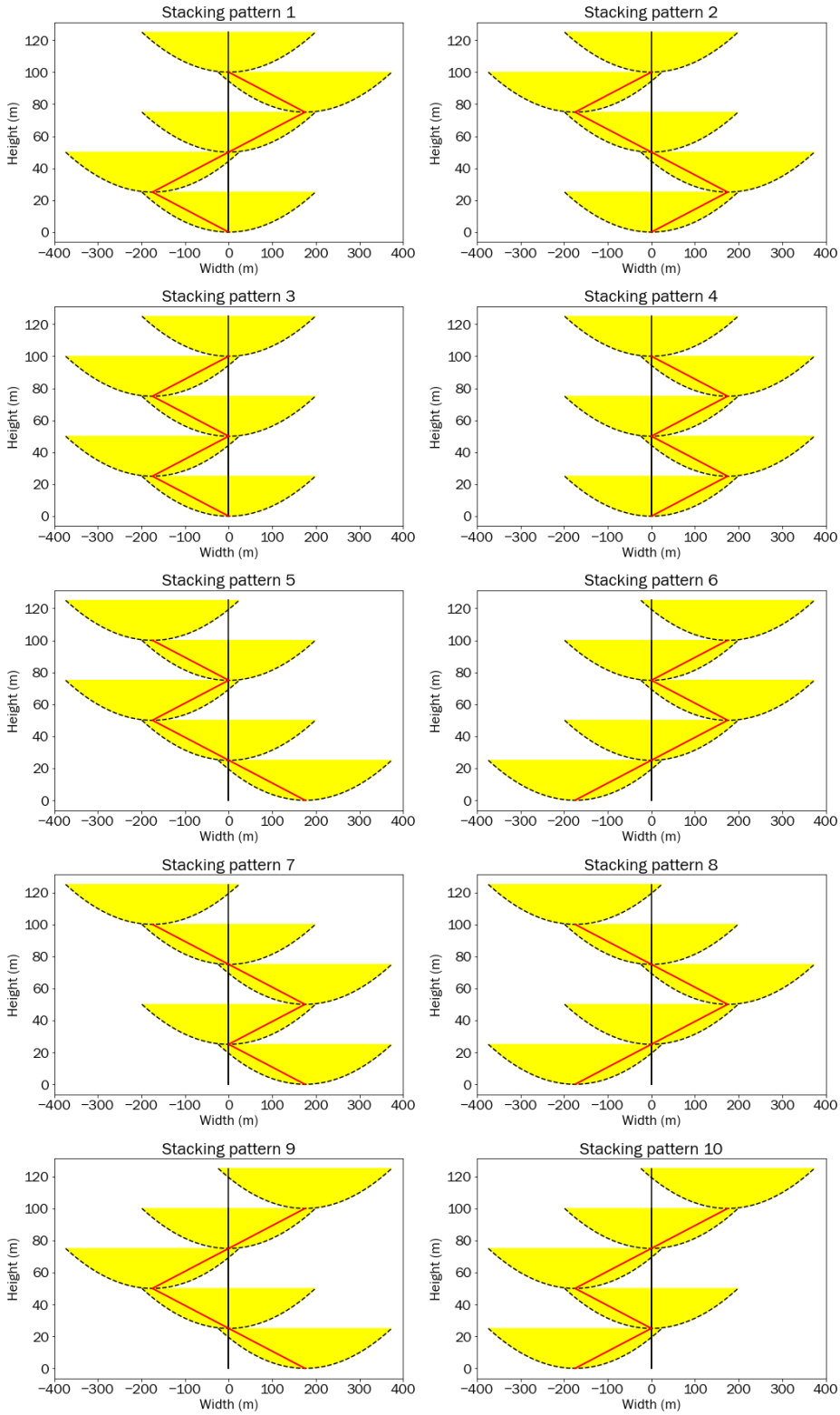


Fig. 37. The 10 most likely channel stacking pattern possibilities for five stacked channel elements using the third hypothetical vertical transition matrix. The channel stacking pattern possibilities are sorted from the one that has more occurrences out of 100000 iterations to the possibility that has fewer occurrences. The black line is the trace of the measured section and the red line follows the parabola vertexes of the channel elements.

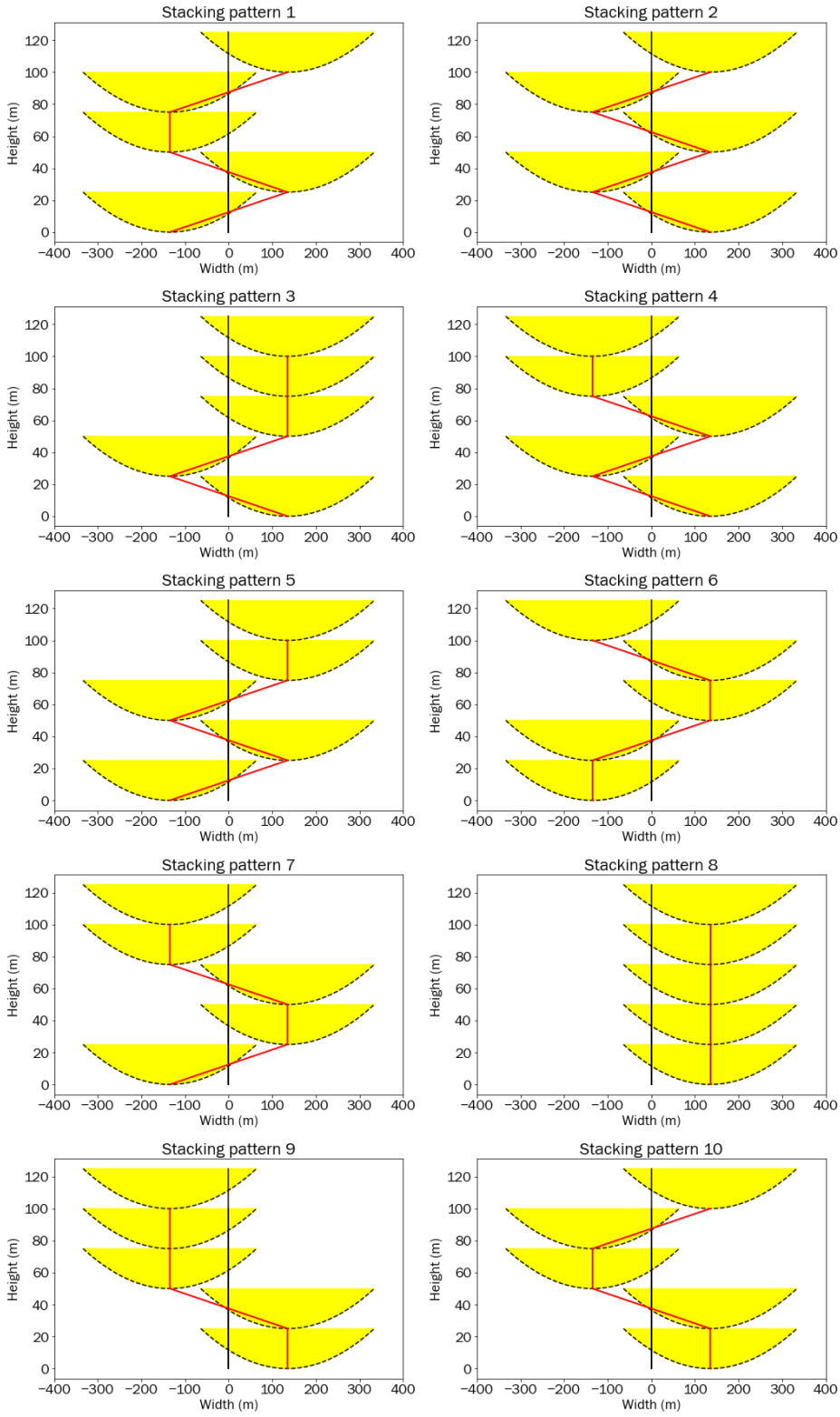


Fig. 38. The 10 least likely channel stacking pattern possibilities for five stacked channel elements using the third hypothetical vertical transition matrix. The channel stacking pattern possibilities are sorted from the one that has more occurrences out of 100000 iterations to the possibility that has fewer occurrences. The black line is the trace of the measured section and the red line follows the parabola vertexes of the channel elements.

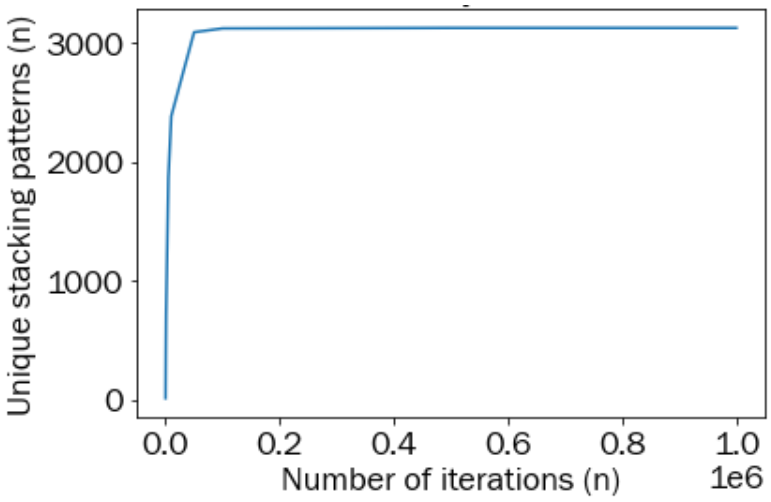


Fig. 39. Variability levels of simulations of channel stacking for the real vertical transition probability matrix (Table 4). The maximum number of iterations is reached after 50000 realizations. The curve was plotted from the results of performing 10, 100, 500, 1000, 5000, 10000, 50000, 100000 and 1000000 iterations per simulation.

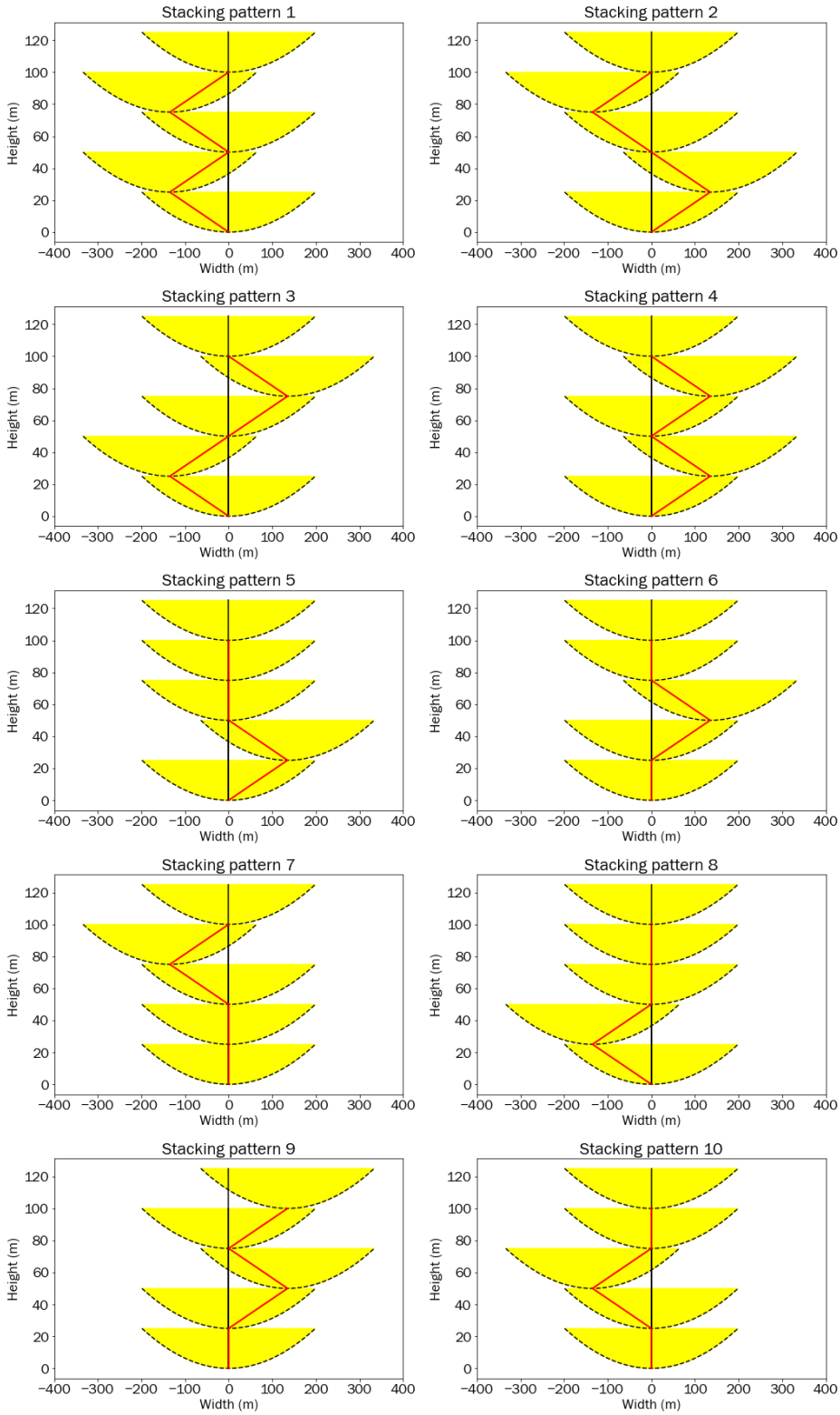


Fig. 40. The 10 most likely channel stacking pattern possibilities for five stacked channel elements using the real vertical transition probability matrix. The channel stacking pattern possibilities are sorted from the one that has more occurrences out of 1000000 iterations to the possibility that has fewer occurrences. The black line is the trace of the measured section and the red line follows the parabola vertexes of the channel elements.

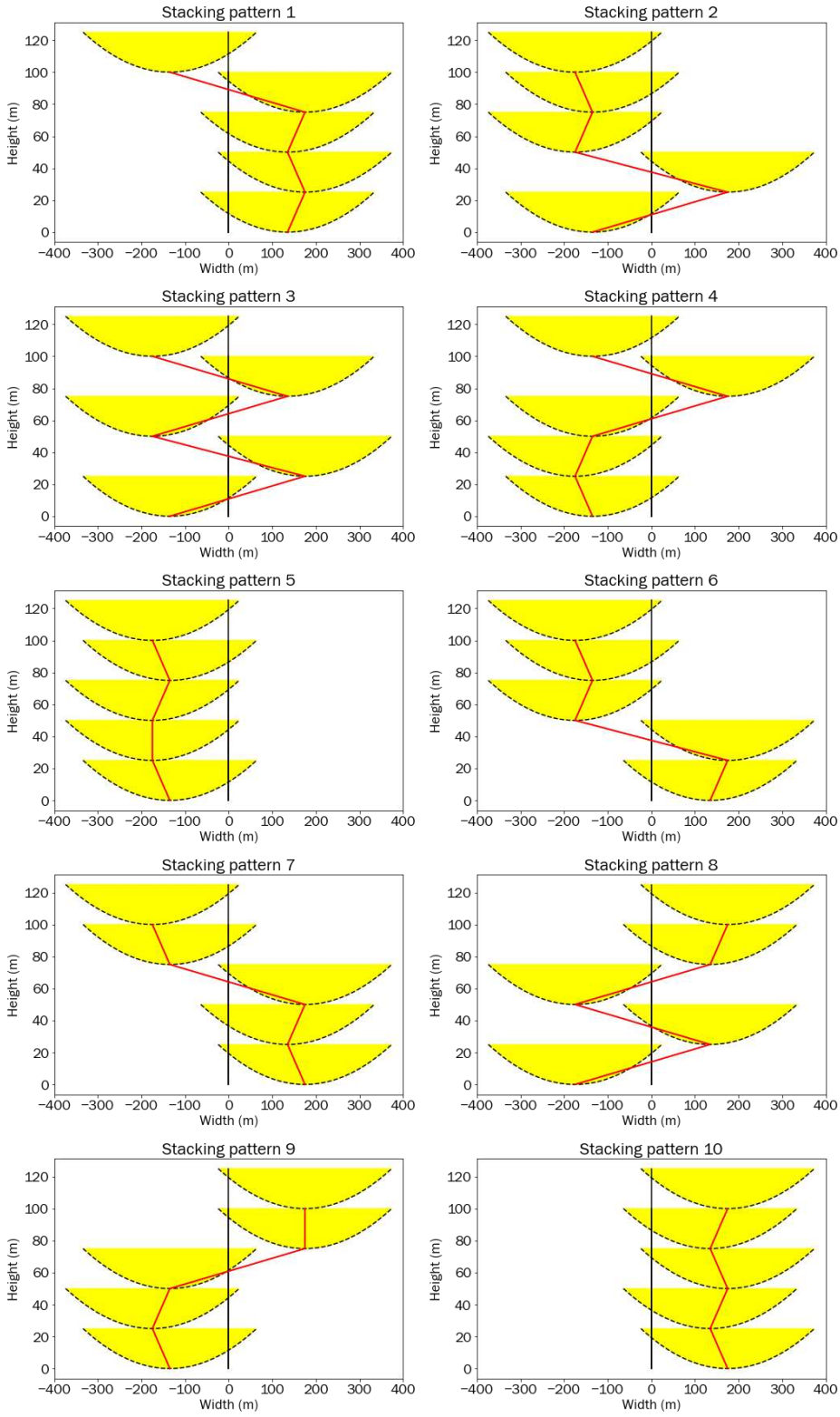


Fig. 41. The 10 least likely channel stacking pattern possibilities for five stacked channel elements using the real vertical transition probability matrix. The channel stacking pattern possibilities are sorted from the one that has more occurrences out of 1000000 iterations to the possibility that has fewer occurrences. The black line is the trace of the measured section and the red line follows the parabola vertexes of the channel elements.

CHAPTER 6. MATCHING CHANNEL STACKING PATTERNS TO THICKNESS

6.1. Methods

6.1.1. Database and coding style

For performing the match to the thickness of the channel stacking patterns, we used a modified version of the machine learning-derived probabilities from the neural network classification of architectural positions of Vento (2020). This includes the thicknesses of each channel element per architectural position (NN_Results_2.xlsx, Appendix A) from the Chile Slope System consortium database (https://www.chileslopesystems.com/). Then, we proceeded to upload this .xlsx file as a pandas.DataFrame in Python using JupyterLab.

This script includes the drawing of the probability density functions (PDF) and cumulative distribution functions (CDF) (Chapter 4) to generate realizations of channel stacking patterns that match the actual thickness of the channel elements within the measured section and therefore, the total thickness.

The script was written in a functional programming style.

6.1.2. Parabola translations

For generating the channel stacking patterns, the code has been dealing with the template of a parabola (Eq. 1). Parameters like width (w), thickness, depth or height (h), lateral offset (x) and vertical offset (y) can be modeled. For this research, width (w) and thickness, depth or height (h) have remained constant.

A parabola can be expressed mathematically by the standard form of the quadratic equation (Eq. 2).

$$y = ax^2 + bx + c \quad (2)$$

A parabola also can be expressed mathematically by the vertex form (Eq. 3), where *a* is a constant or coefficient, *h* is the x-coordinate of the parabola vertex and *k* is the y-coordinate of the parabola vertex.

$$y = a(x - h)^2 + k \quad (3)$$

If the parabola does not have any translation (i.e., shifting), then Eq. 3 is reduced to Eq 4.

$$y = ax^2 \quad (4)$$

And finally, for a constant *x* and *y*, we can calculate the *a* coefficient, or stretching constant (Eq. 5), so the parabolas are going to look the same no matter the translation (*x* and *y* values).

$$a = \frac{y}{x^2} \quad (5)$$

Given that our channel element template is a parabola, the thickness *y* (the distance from the vertex of the parabola to the top) is 25 m and the width (the distance between the minimum and maximum points) in the x-axis is 400 m, and half-width is x = 200 m, the stretching constant for out channel element template *a* is 0.000625.

Now, for populating the channel stacking with the template, and given that a predefined lateral offset (*x*) has been chosen and is related to the architectural positions (axis, off-axis and margin, Chapter 4, section 4.3.4) and have been simulated from the Monte Carlo simulation, the only variable missing is the vertical offset (*y*). It is required to find a list of vertical offsets (*y*) in a way that the parabola translations regarding the *y*-axis match the thickness of the channel elements.

6.2. Results

6.2.1. Obtaining a vertical offset that matches the thickness

Controlling the vertex (h, k) of the parabola (Eq. 3) is a mathematically straightforward procedure to translate a parabola template and the lateral offset (x) has been modeled and is known. Now, we need to find a list of vertical offsets (y) that matches the thicknesses.

Thickness (t) is inversely related to the y-coordinate of the vertex of a parabola k, or height (i.e., when the parabola reaches the maximum thickness, k is 0). Thickness is defined and measured as a perpendicular line that goes from the vertex to the middle top of the parabola. In contrast, k is defined by a perpendicular line from a y-plane at the vertex of a parabola to the base of the parabola. Given that there are only five possible lateral offsets (x), a list of k can be generated to adjust the vertical offset (y) given a certain architectural position (Fig. 42).

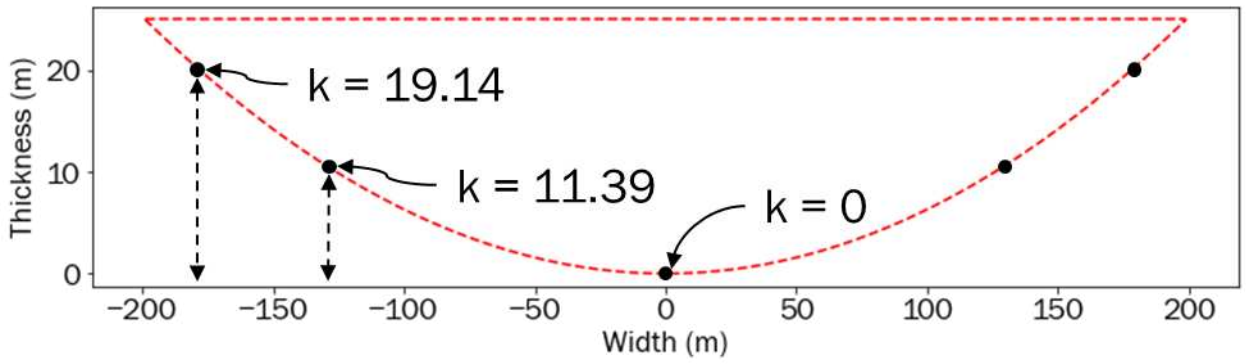


Fig. 42. The three k's used for calculating the final vertical offsets that match thicknesses regarding the architectural position. From lower to greater k, axis, off-axis and margin.

The final list of vertical offsets, or k's that match the thickness, are calculated given that the net vertical offset (y) for a given channel element n is the difference between the cumulative thickness up to channel element n-1 and a k given the channel element n architectural position. The cumulative thickness was also edited so the information can be matched with a measured section at the origin (0, 0) up to (0, y = cumulative thickness).

6.2.2. Channel stacking pattern generation

Then, following the procedure illustrated in Chapter 4 for generating realizations of the channel stacking patterns using the *stacking_patterns* function, it is possible to generate realizations that honor the thickness and the architectural position. A modified version of the *stacking_patterns* function, the *stacking_patterns_to_thick* function, is defined to generate the final channel stacking patterns. Instead of defining a constant vertical offset, it is required to define a list of fixed final vertical offsets.

6.2.3. Channel stacking patterns matched to thickness

From the results of the simulation with 1000000 iterations, we sorted and filtered the 10 most likely channel stacking possibilities (Table 9). The probability of occurrence of those channel stacking patterns is about 1.4% out of 1000000 iterations. Then we plotted the results as channel stacking patterns matched to thickness (Fig. 43).

Furthermore, we created a function that calculates the total distance between the parabola vertexes per channel stacking pattern (*distances* function) as an indicator to compare channel stacking patterns with low vs. high lateral offset (Table 9) matched to thickness. Given this parameter, channel stacking patterns 1, 4 and 9 have less lateral offset regarding the other channel stacking pattern possibilities. This total distance is 355.95 m when matched to thickness. From Fig. 43, those channel stacking patterns appear to be highly vertically aligned.

Following a similar procedure, from the results of the simulation with 1000000 iterations, we sorted and filtered the 10 least likely channel stacking possibilities (Table 10). The probability of occurrence of those channel stacking patterns is 0.0001% out of 1000000 iterations. They are unlikely. Then we plotted the results as channel stacking patterns matched to thickness (Fig. 44).

Table 9. The 10 most likely channel stacking possibilities. Elem: channel element from base to the top, TD: total distance matched to thickness, -2: left margin, -1: left off-axis, 0: axis, 1: right off-axis, 2: right margin.

	Elem1	Elem2	Elem3	Elem4	Elem5	Count	TD
1	-1	0	1	2	1	**14688**	**355.95**
2	-1	0	-1	2	1	**14594**	**625.92**
3	1	0	1	-2	1	**14566**	**892.14**
4	-1	0	-1	-2	-1	**14521**	**355.95**
5	-1	0	1	-2	1	**14509**	**892.14**
6	-1	0	-1	-2	1	**14470**	**622.17**
7	1	0	1	2	-1	**14424**	**622.17**
8	-1	0	1	2	-1	**14355**	**622.17**
9	1	0	1	2	1	**14335**	**355.95**
10	1	0	-1	2	1	**14331**	**625.92**

Table 10. The 10 least likely channel stacking possibilities. Elem: channel element from base to the top, TD: total distance matched to thickness, -2: left margin, -1: left off-axis, 0: axis, 1: right off-axis, 2: right margin.

	Elem1	Elem2	Elem3	Elem4	Elem5	Count	TD
1	0	-1	-2	-2	-2	**1**	**193.86**
2	0	-1	-2	-1	2	**1**	**528.63**
3	1	1	-1	1	-2	**1**	**858.8**
4	-2	2	-2	2	1	**1**	**1094.96**
5	0	-1	-1	-1	2	**1**	**468.93**
6	2	-1	2	1	1	**1**	**674.36**
7	2	-1	2	2	-2	**1**	**976.9**
8	-2	2	-2	1	0	**1**	**1147.77**
9	-2	2	-2	-2	0	**1**	**884.33**
10	2	-2	0	1	1	**1**	**675.48**

Using the total distance between the parabola vertexes per channel stacking pattern as an indicator to compare channel stacking patterns with low vs. high lateral offset (Table 10) matched to thickness, the lateral offset highly varies for these unlikely channel stacking patterns possibilities, from 193.89 (channel stacking pattern 1) to 1147.77 m (channel stacking pattern 8). A little total distance is related to high vertically aligned channel stacking patterns, whereas long distances are related to channel stacking patterns in which the lateral offset between channel elements is greater.

From Fig. 44, off-axis and margin positions are sometimes unrealistically thicker, so there are big gaps between channel elements in some channel stacking pattern possibilities (e.g., channel stacking pattern 10).

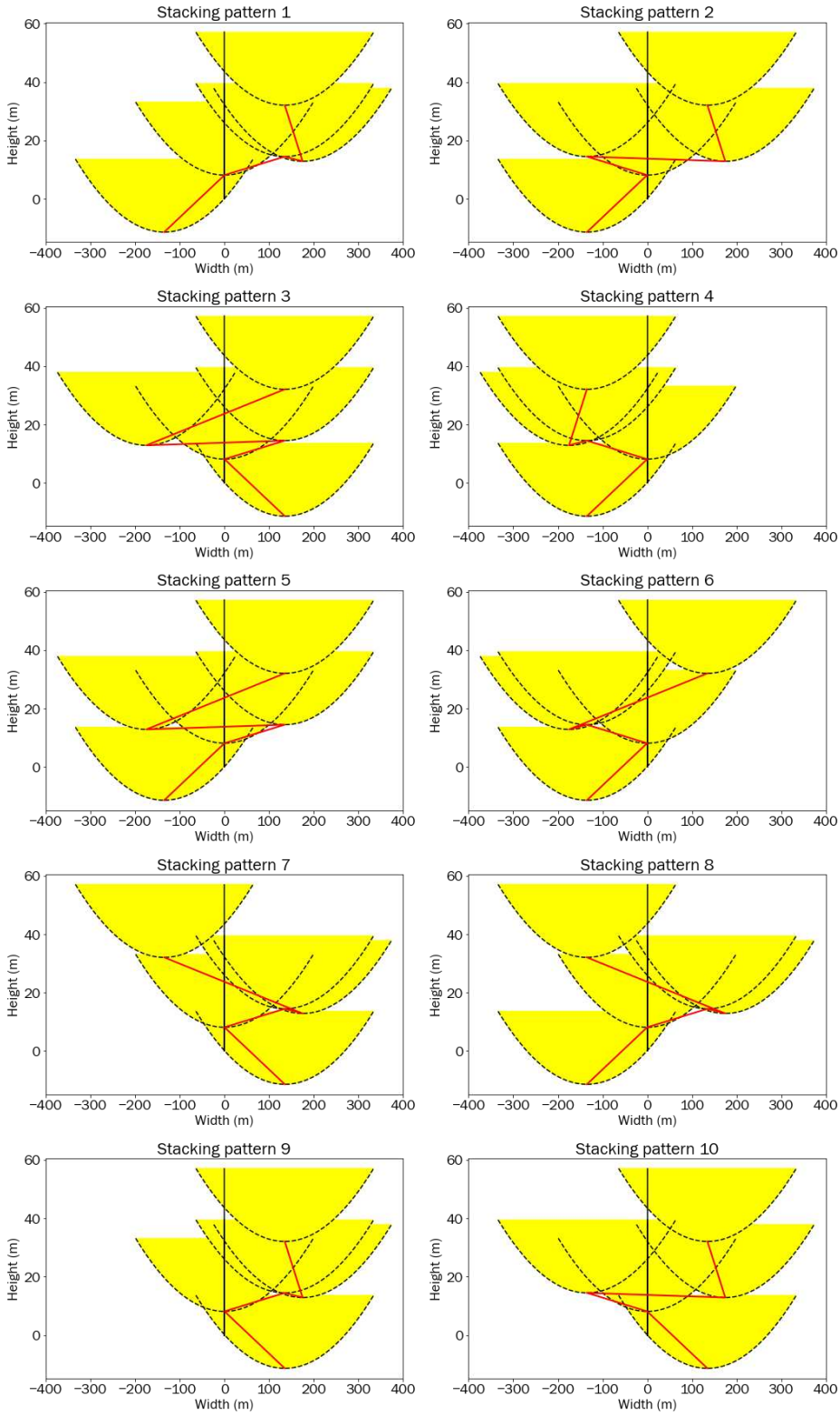


Fig. 43. The 10 most likely channel stacking pattern possibilities for the measured section CACH1, simulating the channel stacking directly from the soft probabilities and matched to thickness. The channel stacking pattern possibilities are sorted from the one that has more occurrences out of 1000000 iterations to the possibility that has fewer occurrences. The black line is the trace of the measured section and the red line follows the parabola vertexes of the channel elements.

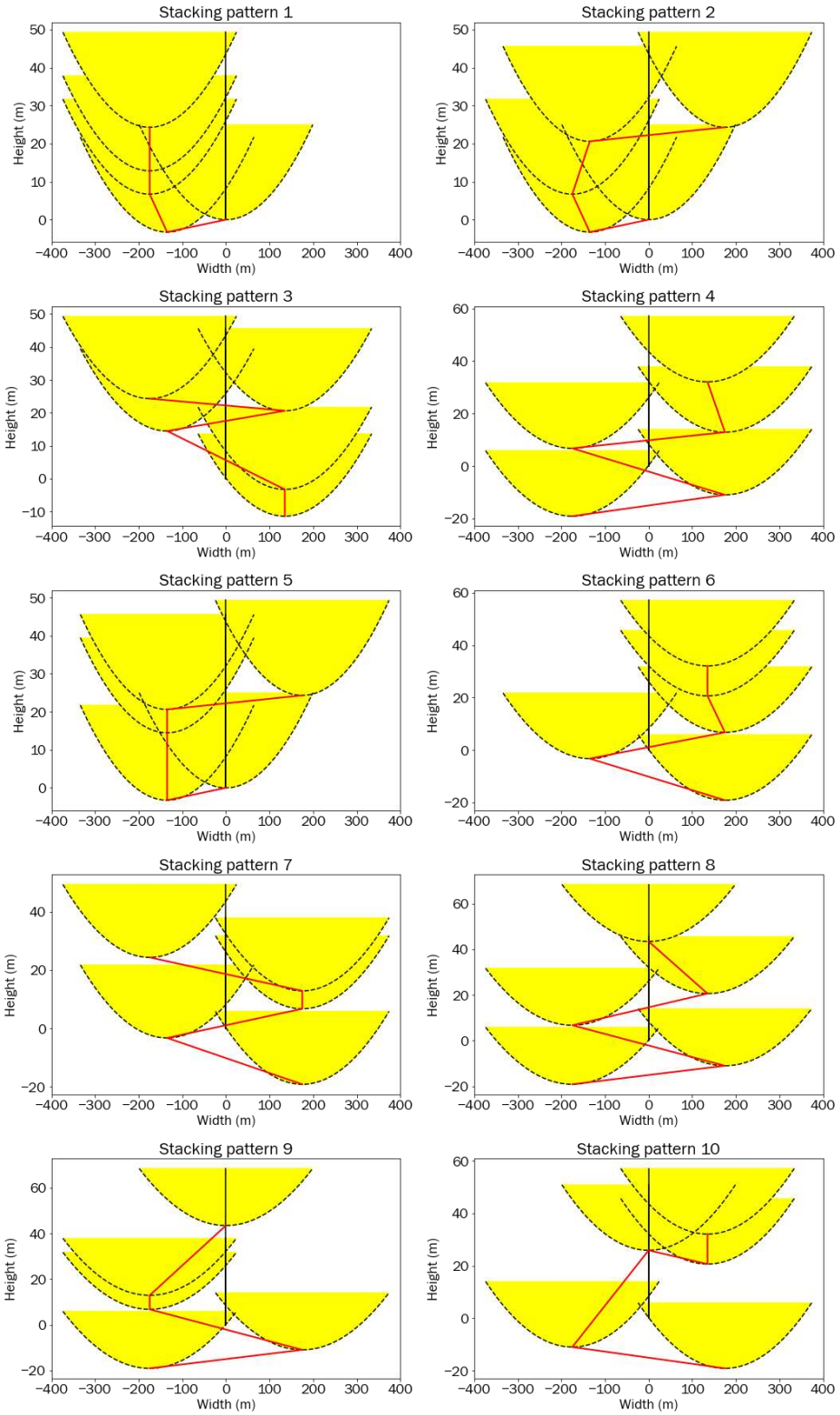


Fig. 44. The 10 least likely channel stacking pattern possibilities for the measured section CACH1, simulating the channel stacking directly from the soft probabilities and matched to thickness. All these channel stacking pattern possibilities had just one occurrence out of 1000000 iterations. The black line is the trace of the measured section and the red line follows the parabola vertexes of the channel elements

7.1. Conditional simulation and soft probabilities

Using machine learning-derived probabilities of classification of architectural positions of channel elements allows us to assess probabilistically, or stochastically, the probability of occurrence of them.

Even though lithological classifications are usually deterministic, the truth is that for sedimentary rocks (and for rocks in general) there is a continuous spectrum between classifications (Fig. 45), and therefore facies and architectural positions. Dealing with the discrete information provided by machine learning techniques as a continuous random variable has more geological sense and provides us with tools to model sedimentary systems.

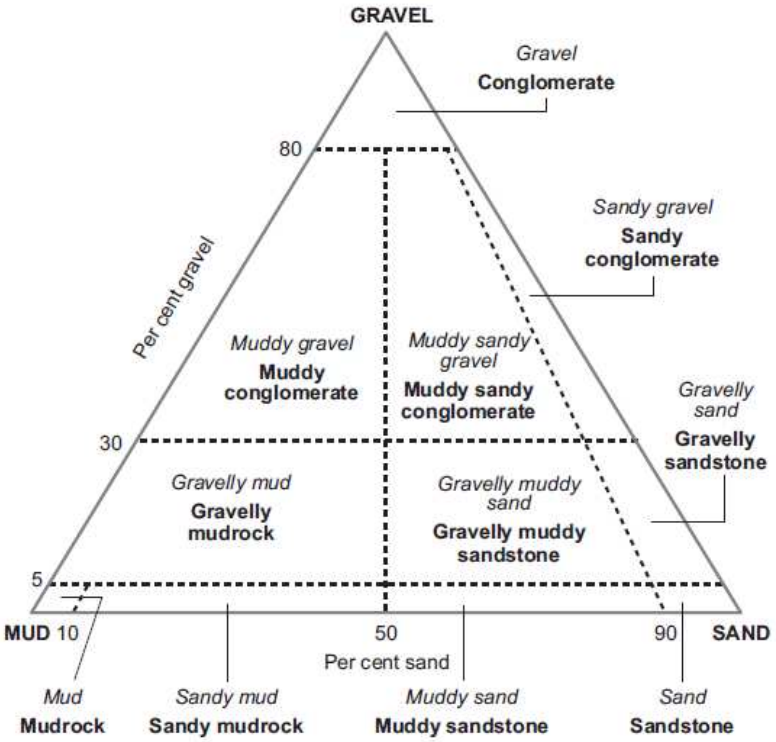


Fig. 45. Nomenclature used for mixtures of gravel, sand and mud in sediments and sedimentary rock (Nichols, 2009).

This approach seems to be successful, we introduced randomness in the modeling of architectural positions. For example, in Fig. 18, even though the probability of occurrence of margin for elements 2 and 5 was virtually 0, after running a Monte Carlo simulation a little probability of occurrence appears. This could be useful for generating stochastic realizations of channel stacking patterns.

Visually, the estimated values from the Monte Carlo simulations are following the distributions of our original dataset, indicating that we are performing a procedure that has statistical ground (Avseth et al., 2005) and also cross-checking that the code is executing correctly.

As laterally contiguous conditions and processes, and then environments of deposition of sediments in a region shift with time in response to geologic conditions (e.g., climate, source areas, tectonism) layers of rocks also shift and eventually the deposits of one depositional environment lie above those of another. This idea constitutes one of the most important concepts in geology. *Walther's Law* establishes that "a direct environmental relationship exists between lateral facies and vertically stacked or superimposed successions of strata" (Boggs, 2014).

In order words, layers of rocks that are found in a vertical profile coexisted horizontally at the same time. Given this, the vertical exhibition of rocks itself should be able to tell something about the horizontal distribution of rocks. This is the fundamental concept behind predicting channel stacking patterns from architectural elements in vertical profiles, and its application in submarine channels is mentioned by Macauley and Hubbard (2013). The second panel of Fig. 19 illustrates this point, showing how it is possible to get a sequence of architectural positions that coexisted horizontally stacked in a vertical profile by the accumulation of sediments at different time steps.

The generation of stacking patterns using five templates related to five lateral offsets that in turn are related to the five (left margin, left off-axis, axis, right off-axis and right margin, Fig. 19) basic architectural positions provided us with a simple framework to generate a baseline of equiprobable channel stacking patterns. Furthermore, using initially a constant vertical offset between channel elements allowed

us to visually assess the “organization” of some channel stacking patterns vs. other ones by enhancing the visualization of them with enough vertical separation (Fig. 22, Fig. 23 and Fig. 24). Also, it would allow us to build models for assessing theoretical scenarios in which the deposition of sediments was constant over cyclic time intervals.

Modeling the architectural positions of individual channel elements with conditional simulation and soft probabilities using Monte Carlo simulation and then using the most likely channel stacking to generate realizations of the 2D channel stacking patterns, it was possible to get equiprobable realizations of channel stacking patterns.

Applying the qualitative, predictive rules provides us with insight into which channel stacking pattern is more likely. The most important are:

- Channel elements tend to stack with lateral offsets of less than a channel width in channel complexes with organized channel stacking (Macauley and Hubbard, 2013; McHargue et al., 2011a).
- Within individual submarine channel systems, channel size variability is relatively small, meaning that channel elements within a channel complex tend to have a characteristic and relatively constant size and shape (Sylvester et al., 2011).
- Channels migrate systematically and their placement is not random in organized channel stacking (Sylvester et al., 2011).

As mentioned, the Tres Pasos Fm. channels have been considered as showing an organized stacking pattern (Macauley and Hubbard, 2013). From equiprobable realizations, the prediction will depend on the background, knowledge and experience of the interpreter. Nevertheless, applying the predictive rules and geological thinking, from Fig. 25 the most likely channel stacking patterns are numbers 8 and 9 (Fig. 46).

The specific selection from those two can be done by leveraging seismic attributes, or seismic probability cubes (Langenkamp, 2021) and observations from partial tridimensional perspectives in the field (Hubbard et al., 2023).

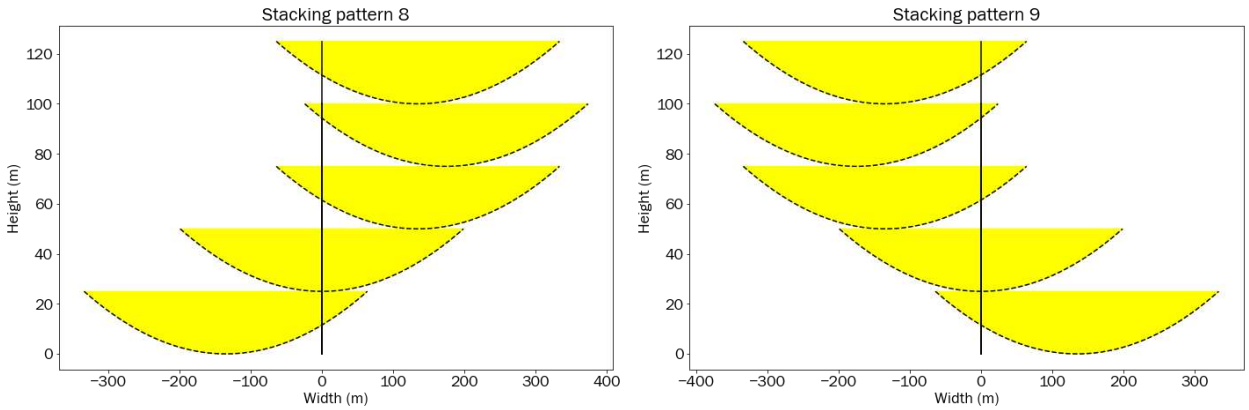


Fig. 46. Qualitatively the most likely channel stacking patterns with independent probabilities and Monte Carlo simulation. A constant vertical offset of 25 m was used.

For quantifying the probability of occurrence of the channel stacking patterns given the conditional simulations with soft probabilities, we focused on modeling channel stacking (instead of modeling individual channel elements). After performing experiments with several millions of iterations, it was not possible to get the total number of possible channel stacking patterns given the soft probabilities. The probabilities of classification for axis and margin from the neural network (Vento, 2020) are high (around 90% of accuracy), so generating the total number of channel stacking patterns is itself unlikely. The curve from Fig. 26 started to get asymptotic by 1000000 realizations, or 2000 channel stacking patterns. The least likely channel stacking possibilities are unrealistic stacking patterns that do not match the measured section data.

7.2. Forward modeling with Markov transition probabilities

We performed forward modeling or automatic channel stacking pattern construction from a seed with Markov transitional probabilities, in this case, the architectural position of the first channel element of a vertical profile. Several hypothetical vertical transition probability matrices behaved as expected. Using the real vertical transition probability matrix, only the axis and off-axis positions stack vertically, reflecting that many transitional pairs counted are from axis to off-axis, from off-axis to off-axis and from off-axis to axis. This could be a result of measuring vertical profiles preferentially through sandy bodies, the targets of the industry. One channel stacking pattern was found to be the most likely vertically, by 0.4%. Nevertheless, by stacking channel elements using just the Markov transition probabilities, the channels have a significant horizontal offset between them, and the channel stacking pattern appears disorganized, in sharp contrast with the products seen in nature and expected using qualitative or predictive rules (Macauley and Hubbard, 2013; McHargue et al., 2011a, 2011b; Sylvester et al., 2011).

Given that, for using transition probabilities matrices for controlling the occurrence of channel stacking patterns you would need to incorporate either (i) a distribution regarding the lateral offset or (ii) consider the right and left architectural positions within the vertical transition probability matrix.

The total distance metric used to evaluate the organization of channel stacking patterns (*sensu* Langenkamp, 2021) provides insight into the lateral offset, and it was taken in both channel stacking patterns with a constant vertical offset and channel stacking patterns matched to thickness. It was useful in both scenarios, particularly in the matched-to-thickness one when you cannot see the distribution of channel elements within the stacking pattern. We can now choose quantitatively which stacking pattern would be more likely. The most likely possibilities usually are equiprobable and have the same total distance or less dispersion. The total distances of least likely possibilities have greater dispersion.

7.3. Matching to thickness

Models of submarine channels are usually generated from seismic data alone (Ringrose and Bentley, 2021). Geoscientists must study seismic-scale outcrops bridging the gap between the aerial extent

of seismic data and the characterization of stratigraphy (Arnott, 2010) including architecture and channel stacking patterns. However, finding those outcrops is rare, and that is why the study of the submarine channels of the Tres Pasos Fm. offers a unique opportunity to contribute to solving a variety of geological problems.

As mentioned, previous studies have focused on understanding submarine channels using statistics and numerical models (Li and Caers, 2011; McHargue et al., 2011a, 2011b; Morris et al., 2022; Rongier et al., 2017; Sylvester et al., 2011) with the rare exception of Macauley and Hubbard (2013), that used outcrops. The study of outcrops is a crucial component of exploration geology and ranks as the most important source of information from the Earth and the one that should be honored the most over indirect surveys (well logs and seismic data). This research attempted to bridge the gap between computational models and information provided by outcrops.

For doing that, matching thickness has been recognized as one of the most challenging tasks (Li and Caers, 2011). Previous authors were able to achieve it using a complex series of operations in grids/rasters (Li and Caers, 2011), two-step simulations (Pyrcz et al., 2015) and artificial neural networks (Titus et al., 2021). Nevertheless, information from outcrops was not considered, and in this research matching thickness was accomplished using information from outcrops for the very first time.

7.4. Future work

Jackson et al. (2019) proved that channel stacking patterns impact connectivity, and that high lateral channel element offsets reduce it. Assessing multiple channel stacking patterns is crucial for understanding flow units and has implications on well placement and reservoir connectivity. Geological interpretations are often very qualitative and deterministic, so the results of this research provide a framework to evaluate stochastically interpretations and can even be extended to the assessment of uncertainty and risk of subseismic architecture and its impact on reservoir connectivity. Furthermore, adding more stratigraphic

complexity, the impacts of thin bed presence and position could be stochastically evaluated when having channel stacking patterns with different lateral or vertical offsets (Meirovitz et al., 2020).

Even though our models are simple, they have the potential to give clues for better modeling submarine channels while honoring the data and using statistical techniques. Given the lack of "true" or "ground" information, by replicating the methods of this research to the other measured sections in the Tres Pasos Fm. and adding some stratigraphic complexities (like the occurrence of mass transport deposits, MTDs), the deterministic interpretations of channel stacking patterns could be changed in a minimum time while recognizing uncertainty. This would allow to revisit the planform deterministic interpretations of Macauley and Hubbard (2013) and strengthen the statistics of the submarine channels of the Tres Pasos Fm.

Currently, the advent of new methodologies to build predictive reservoir models of the subsurface like rule-, surface- or process-based modeling allows us to obtain clues about the evolution of channelized systems (Pyrcz et al., 2015). However, these models rely on algorithms for centerline generation adapted from fluvial models (Covault et al., 2016; McHargue et al., 2011b, 2011a; Morris et al., 2022; Pyrcz et al., 2015; Sylvester et al., 2011).

Despite submarine channels and fluvial channels share features like the development of terraces and channel morphologies, from braided to meandering, (i) submarine channels are larger and occur more commonly than subaerial channels, (ii) mainly intermittent turbidity currents construct submarine channels, (iii) those flows travel more slowly than an equidimensional river and (iv) Coriolis effects are more significant for turbidity currents than for fluvial streams (MacNaughton et al., 2005).

By extrapolating the results of this research to the other measured sections would be possible to define paleochannel trajectory points per measured section that match the "ground" data, and by using Non-Uniform Rational B-Splines (NURBS) (Jacquemyn et al., 2019; Ruiu et al., 2016) find the paleochannel trajectory and build algorithms for centerline generation that considers the difference between fluvial and

submarine channels. Also, it would be possible to analyze the paleotopography of the channelized surface and its influence on depositional and erosional processes and the preservation of architecture.

CHAPTER 8. CONCLUSIONS

Architectural position probabilities generated by machine learning are a useful foundation for generating near-wellbore models of channel stacking, and modeling stochastically a discrete random variable like architectural position allows to assess the probability of occurrence while making more sense of the continuous nature of sedimentary systems. We demonstrated a method to generate multiple scenarios of channel stacking patterns at measured sections using templates of channel elements. Fixed lateral offsets depict different architectural positions.

Conditional Monte Carlo simulation with soft probabilities per channel element allowed us to build equiprobable realizations of channel stacking patterns but are unable to quantify their probability of occurrence.

Modeling channel stacking (instead of modeling individual channel elements) allowed us to quantify the probability of occurrence of the possibilities. Impossible channel stacking patterns are unlikely to be obtained given the high probabilities of occurrence of the classification of architectural position from the soft probabilities. Using forward modeling or Markov transitional probabilities for automatic channel stacking pattern construction from a seed without constraints from the measured section yields slightly more likely channel stacking patterns but unrealistic lateral offsets. The total distance metric provides insight into the organization and can be used to compare low vs. high lateral offsets scenarios.

The channel stacking patterns generated match thickness using a straightforward procedure that can be replicated without the use of time-expensive, complex simulations. Deterministic interpretations of channel stacking patterns can be revisited in a minimum time.

REFERENCES

Arnott, R.W.C., 2010. Deep-marine sediments and sedimentary systems, in: Facies Models. pp. 295–322.

Avseth, P., Mukerji, T., Mavko, G., 2005. Quantitative seismic interpretation: Applying rock physics tools to reduce interpretation risk, Quantitative Seismic Interpretation: Applying Rock Physics Tools to Reduce Interpretation Risk. https://doi.org/10.1017/CBO9780511600074

Boggs, S., 2014. Principles of sedimentology and stratigraphy, Fifth edit. ed, Sedimentary Geology. Pearson. https://doi.org/10.1016/0037-0738(95)00151-4

Cao, W., Zhou, A., Shen, S.L., 2021. An analytical method for estimating horizontal transition probability matrix of coupled Markov chain for simulating geological uncertainty. Computers and Geotechnics 129, 103871. https://doi.org/10.1016/j.compgeo.2020.103871

Covault, J.A., Sylvester, Z., Hubbard, S.M., Jobe, Z.R., Sech, R.P., 2016. The Stratigraphic Record of Submarine-Channel Evolution. The Sedimentary Record 14, 4–11. https://doi.org/10.2110/sedred.2016.3.4

Daniels, B.G., Auchter, N.C., Hubbard, S.M., Romans, B.W., Matthews, W.A., Stright, L., 2018. Timing of deep-water slope evolution constrained by large-n detrital and volcanic ash zircon geochronology, Cretaceous Magallanes Basin, Chile. Bulletin of the Geological Society of America 130, 438–454. https://doi.org/10.1130/B31757.1

Elfeki, A., Dekking, M., 2001. A Markov chain model for subsurface characterization: Theory and applications. Mathematical Geology 33, 569–589. https://doi.org/10.1007/s11004-006-9037-9

Fletcher, S., 2013. Stratigraphic Characterization of a Cretaceous Slope Channel Complex in the Tres Pasos Formation, Arroyo Picana-Laguna Figueroa Outcrop Belt, Chilean Patagonia. https://doi.org/10.11575/PRISM/27886

Fosdick, J.C., Romans, B.W., Fildani, A., Bernhardt, A., Calderón, M., Graham, S.A., 2011. Kinematic evolution of the Patagonian retroarc fold-and-thrust belt and Magallanes foreland basin, Chile and Argentina, 51°30's. Bulletin of the Geological Society of America 123, 1679–1698. https://doi.org/10.1130/B30242.1

Gómez, J., Schobbenhaus, C., Montes, N.E., 2019. Geological Map of South America 2019. Scale 1:5 000 000. Commission for the Geological Map of the World (CGMW), Colombian Geological Survey and Geological Survey of Brazil, Paris. https://doi.org/https://doi.org/10.32685/10.143.2019.92

Heijnen, M.S., Clare, M.A., Cartigny, M.J.B., Talling, P.J., Hage, S., Pope, E.L., Bailey, L., Sumner, E., Gwyn Lintern, D., Stacey, C., Parsons, D.R., Simmons, S.M., Chen, Y., Hubbard, S.M., Eggenhuisen, J.T., Kane, I., Hughes Clarke, J.E., 2022. Fill, flush or shuffle: How is sediment carried through submarine channels to build lobes? Earth and Planetary Science Letters 584, 117481. https://doi.org/10.1016/j.epsl.2022.117481

Hubbard, S.M., Covault, J.A., Fildani, A., Romans, B.W., 2014. Sediment transfer and deposition in slope channels: Deciphering the record of enigmatic deep-sea processes from outcrop. Bulletin of the Geological Society of America 126, 857–871. https://doi.org/10.1130/B30996.1

Hubbard, S.M., Romans, B.W., Southern, S., Stright, L., Daniels, B.G., Fletcher, S.A., Jackson, A., Kaempfe, S., Macauley, R. V., Nielsen, A., Niquet, D., Mierowitz, C., Pemberton, E.A.L., Reimchen, A., 2018. Core- and log-based recognition criteria for deep-water channel bodies: Using outcrops to inform stratigraphic architecture predictions beyond the wellbore. AAPG Annual Convention and Exhibition, Salt Lake City, Utah, May 20-23, 2018.

Hubbard, S.M., Romans, B.W., Stright, L., Kaempfe, S., 2023. Stratigraphic Architecture and Evolution of Slope Systems: Magallanes Basin, southern Chile. Calgary.

Jackson, A., Stright, L., Hubbard, S.M., Romans, B.W., 2019. Static connectivity of stacked deep-water channel elements constrained by high-resolution digital outcrop models. AAPG Bulletin 103, 2943–

2973. https://doi.org/10.1306/03061917346

Jackson, M.D., Hampson, G.J., Saunders, J.H., El-Sheikh, A., Graham, G.H., Massart, B.Y.G., 2014. Surface-based reservoir modelling for flow simulation. Geological Society, London, Special Publications 387, 271–292. https://doi.org/10.1144/sp387.2

Jacquemyn, C., Jackson, M.D., Hampson, G.J., 2019. Surface-Based Geological Reservoir Modelling Using Grid-Free NURBS Curves and Surfaces. Mathematical Geosciences 51, 1–28. https://doi.org/10.1007/s11004-018-9764-8

Jobe, Z.R., Howes, N.C., Auchter, N.C., 2016. Comparing submarine and fluvial channel kinematics: Implications for stratigraphic architecture. Geology 44, 931–934. https://doi.org/10.1130/G38158.1

Krumbein, W.C., Dacey, M.F., 1969. Markov chains and embedded Markov chains in geology. Journal of the International Association for Mathematical Geology 1, 79–96. https://doi.org/10.1007/BF02047072

Langenkamp, T.R., 2021. Evaluating the impact of deep-water channel architecture on the probability of correct facies classification using 3D synthetic seismic data. Colorado State University.

Li, H., Caers, J., 2011. Geological Modelling and History Matching of Multi-Scale Flow Barriers in Channelized Reservoirs: Methodology and Application. Petroleum Geoscience 17, 17–34. https://doi.org/10.1144/1354-079309-825

Macauley, R. V., Hubbard, S.M., 2013. Slope channel sedimentary processes and stratigraphic stacking, Cretaceous Tres Pasos Formation slope system, Chilean Patagonia. Marine and Petroleum Geology 41, 146–162. https://doi.org/10.1016/j.marpetgeo.2012.02.004

MacNaughton, R., Beauchamp, B., Dewing, K., Smith, I., Wilson, N., Zonneveld, J.P., 2005. Encyclopedia of sediments and sedimentary rocks. Geoscience Canada.

McHargue, T., Pyrcz, M.J., Sullivan, M.D., Clark, J., Fildani, A., Levy, M., Drinkwater, N., Posamentier,

H., Romans, B., Covault, J., 2011a. Event-Based Modeling of Turbidite Channel Fill, Channel Stacking Pattern, and Net Sand Volume. Outcrops Revitalized 163–173. https://doi.org/10.2110/sepmcsp.10.163

McHargue, T., Pyrcz, M.J., Sullivan, M.D., Clark, J.D., Fildani, A., Romans, B.W., Covault, J.A., Levy, M., Posamentier, H.W., Drinkwater, N.J., 2011b. Architecture of turbidite channel systems on the continental slope: Patterns and predictions. Marine and Petroleum Geology 28, 728–743. https://doi.org/10.1016/j.marpetgeo.2010.07.008

Meirovitz, C.D., Stright, L., Hubbard, S.M., Romans, B.W., 2020. The influence of inter-and intra-channel architecture on deep-water turbidite reservoir performance. Petroleum Geoscience 27. https://doi.org/10.1144/petgeo2020-005

Morris, P.D., Sylvester, Z., Covault, J.A., Mohrig, D., 2022. Channel trajectories control deep-water stratigraphic architecture. Depositional Record 880–894. https://doi.org/10.1002/dep2.189

Mpodozis, C., Mella, P., Padva, D., 2011. Estratigrafía y megasecuencias sedimentarias en la cuenca Austral – Magallanes , Argentina y Chile. VIII Congreso de Exploración y Desarrollo de Hidrocarburos 97–138.

Nichols, G., 2009. Sedimentology and stratigraphy, Second edi. ed. Wiley-Blackwell.

Nordahl, K., Ringrose, P.S., Wen, R., 2005. Petrophysical characterization of a heterolithic tidal reservoir interval using a process-based modelling tool. Petroleum Geoscience 11, 17–28. https://doi.org/10.1144/1354-079303-613

Pyrcz, M.J., Boisvert, J.B., Deutsch, C. V., 2009. ALLUVSIM: A program for event-based stochastic modeling of fluvial depositional systems. mComputers and Geosciences 35, 1671–1685. https://doi.org/10.1016/j.cageo.2008.09.012

Pyrcz, M.J., Catuneanu, O., Deutsch, C. V., 2005. Stochastic surface-based modeling of turbidite lobes.

American Association of Petroleum Geologists Bulletin 89, 177–191. https://doi.org/10.1306/09220403112

Pyrcz, M.J., Deutsch, C. V., 2014. Geostatistical reservoir modeling. Oxford University Press.

Pyrcz, M.J., Sech, R.P., Covault, J.A., Willis, B.J., 2015. Stratigraphic rule-based reservoir modeling. Bulletin of Canadian Petroleum Geology 63, 287–303.

Qi, X.H., Li, D.Q., Phoon, K.K., Cao, Z.J., Tang, X.S., 2016. Simulation of geologic uncertainty using coupled Markov chain. Engineering Geology 207, 129–140. https://doi.org/10.1016/j.enggeo.2016.04.017

Remy, N., Boucher, A., Wu, J., 2009. Applied geostatistics with SGeMS, First edit. ed. Cambridge University Press, New York.

Ringrose, P.S., Bentley, M., 2021. Reservoir Model Design, Second. ed, Reservoir Model Design. Springer. https://doi.org/10.1007/978-94-007-5497-3

Romans, B.W., Fildani, A., Hubbard, S.M., Covault, J.A., Fosdick, J.C., Graham, S.A., 2011. Evolution of deep-water stratigraphic architecture, Magallanes Basin, Chile. Marine and Petroleum Geology 28, 612–628. https://doi.org/10.1016/j.marpetgeo.2010.05.002

Romans, B.W., Hubbard, S.M., Graham, S.A., 2009. Stratigraphic evolution of an outcropping continental slope system, Tres Pasos Formation at Cerro Divisadero, Chile. Sedimentology 56, 737–764. https://doi.org/10.1111/j.1365-3091.2008.00995.x

Rongier, G., Collon, P., Renard, P., 2017. A geostatistical approach to the simulation of stacked channels. Marine and Petroleum Geology 82, 318–335. https://doi.org/10.1016/j.marpetgeo.2017.01.027

Ruetten, A., 2021. Evaluating the Impact of Hierarchical Deep-Water Slope Channel Architecture on Fluid Flow Behavior, Cretaceous Tres Pasos Formation, Chile. Colorado State University.

Ruiu, J., Caumon, G., Viseur, S., 2016. Modeling Channel Forms and Related Sedimentary Objects Using a Boundary Representation Based on Non-uniform Rational B-Splines. Mathematical Geosciences 48, 259–284. https://doi.org/10.1007/s11004-015-9629-3

Southern, S., Stright, L., Jobe, Z., Romans, B., Hubbard, S., 2017. The Stratigraphic Expression of Slope Channel Evolution: Insights From Qualitative and Quantitative Assessment of Channel Fills From the Cretaceous Tres Pasos Formation, Southern Chile.

Stright, L., Bernhardt, A., Boucher, A., 2013. DFTopoSim: Modeling Topographically-Controlled Deposition of Subseismic Scale Sandstone Packages Within a Mass Transport Dominated Deep-Water Channel Belt. Mathematical Geosciences 45, 277–296. https://doi.org/10.1007/s11004-013-9444-7

Sylvester, Z., Pirmez, C., Cantelli, A., 2011. A model of submarine channel-levee evolution based on channel trajectories: Implications for stratigraphic architecture. Marine and Petroleum Geology 28, 716–727. https://doi.org/10.1016/j.marpetgeo.2010.05.012

Titus, Z., Heaney, C., Jacquemyn, C., Salinas, P., Jackson, M.D., Pain, C., 2021. Conditioning surface-based geological models to well data using artificial neural networks. Computational Geosciences. https://doi.org/10.1007/s10596-021-10088-5

Vento, N.F.R., 2020. Hypothesis-based machine learning for deep-water channel systems. Colorado State University.

Zhang, X., Pyrcz, M.J., Deutsch, C. V., 2009. Stochastic surface modeling of deepwater depositional systems for improved reservoir models. Journal of Petroleum Science and Engineering 68, 118–134. https://doi.org/10.1016/j.petrol.2009.06.019

APPENDIX A – DATABASE

- NN_Results_2.xlsx

Axis_Prob	Off_Axis_Prob	Margin_Prob	Pred_Class	True_Class	Geobody	Meas_Sect	Complex Set	Thickness
0.20	0.76	0.04	2	2	2	CACH1	'Lower'	8.13
0.90	0.10	0.00	1	1	3	CACH1	'Lower'	17.73
0.19	0.48	0.33	2	2	6	CACH1	'Lower'	6.14
0.00	0.10	0.90	3	3	7	CACH1	'Lower'	11.45
0.22	0.78	0.00	2	2	8	CACH1	'Lower'	18.39
0.01	0.17	0.83	3	3	2	CACH2	'Lower'	6.61
0.73	0.27	0.00	1	1	3	CACH2	'Lower'	20.10
0.00	0.00	1.00	3	3	8	DS1	'Lower'	6.58
0.00	0.00	1.00	3	3	9	DS1	'Lower'	6.14
0.00	0.21	0.79	3	3	3	DYMD1	'Lower'	7.84
0.25	0.55	0.20	2	2	2	FIG100	'Upper'	8.05
0.00	0.10	0.90	3	3	3	FIG100	'Upper'	10.02
0.99	0.01	0.00	1	1	8	FIG100	'Upper'	21.85
0.88	0.11	0.00	1	2	10	FIG100	'Upper'	11.84
0.00	0.09	0.91	3	3	11	FIG100	'Upper'	13.97
0.00	0.08	0.91	3	3	12	GC1	'Upper'	10.38
0.48	0.51	0.01	2	1	11	GC10	'Upper'	10.24
0.00	0.10	0.90	3	3	12	GC2	'Upper'	10.28
0.81	0.19	0.00	1	1	12	GC3	'Upper'	19.20
0.29	0.70	0.00	2	2	12	GC4	'Upper'	16.71

0.88	0.12	0.00	1	1	11	GC6	'Upper'	10.79
0.67	0.33	0.00	1	1	12	GC7	'Upper'	24.00
1.00	0.00	0.00	1	1	12	GC8	'Upper'	24.40
0.00	0.02	0.98	3	3	12	GCNOR	'Upper'	6.75
0.99	0.01	0.00	1	1	8	GD1	'Upper'	11.59
0.40	0.59	0.00	2	2	10	GD1	'Upper'	12.50
0.00	0.08	0.92	3	3	2	GD3	'Upper'	5.22
0.03	0.28	0.70	3	3	5	GD3	'Upper'	7.62
0.15	0.60	0.25	2	1	10	GD5GC5	'Upper'	6.42
0.85	0.12	0.03	1	1	11	GD5GC5	'Upper'	6.31
0.99	0.01	0.00	1	1	12	GD5GC5	'Upper'	23.56
0.02	0.47	0.51	3	3	6	GD6KS7	'Upper'	7.24
0.66	0.33	0.01	1	1	11	GD6KS7	'Upper'	7.32
0.97	0.03	0.00	1	1	11	GD7	'Upper'	11.79
0.00	0.01	0.99	3	3	5	GD8	'Upper'	11.74
0.00	0.01	0.99	3	3	3	GOLDA	'Lower'	8.32
0.04	0.75	0.21	2	2	3	GOLDB	'Lower'	10.78
0.09	0.82	0.09	2	2	3	GOLDC	'Lower'	11.30
0.11	0.88	0.01	2	2	3	GOLDD	'Lower'	15.48
0.77	0.23	0.00	1	1	2	KJ1	'Lower'	9.18
0.11	0.87	0.02	2	2	3	KJ1	'Lower'	13.87
0.04	0.70	0.26	2	2	5	KJ1	'Lower'	11.27
0.98	0.02	0.00	1	1	7	KJ1	'Lower'	7.94
0.00	0.23	0.77	3	3	8	KJ1	'Lower'	7.72
0.01	0.92	0.08	2	2	9	KJ1	'Lower'	12.33
0.00	0.06	0.94	3	3	10	KJ1	'Lower'	6.15
0.00	0.00	1.00	3	3	12	KS3	'Upper'	8.52
0.95	0.05	0.00	1	1	7	MM1	'Lower'	12.86

0.00	0.00	1.00	3	3	3	MM101	'Lower'	6.34
0.41	0.57	0.02	2	2	5	MM101	'Lower'	9.86
0.47	0.53	0.00	2	1	7	MM101	'Lower'	13.07
0.15	0.85	0.00	2	1	9	MM101	'Lower'	22.26
0.00	0.09	0.91	3	3	10	MM101	'Lower'	4.86
0.06	0.94	0.00	2	2	2	MM102	'Lower'	15.90
0.00	0.01	0.99	3	3	3	MM102	'Lower'	6.30
0.32	0.67	0.00	2	2	5	MM102	'Lower'	10.40
0.02	0.84	0.13	2	2	6	MM102	'Lower'	9.20
0.27	0.69	0.04	2	2	7	MM102	'Lower'	11.32
0.82	0.18	0.00	1	1	9	MM102	'Lower'	21.51
0.05	0.76	0.19	2	2	5	MM103	'Upper'	9.66
0.77	0.23	0.00	1	1	12	MM103	'Upper'	22.76
0.00	0.00	1.00	3	3	7	MM2	'Lower'	9.08
0.05	0.60	0.35	2	2	9	OP1	'Lower'	7.80
0.27	0.71	0.02	2	2	6	OP2	'Lower'	13.09
0.85	0.15	0.00	1	1	8	OP2	'Lower'	25.78
0.13	0.65	0.22	2	2	9	OP2	'Lower'	7.09
0.24	0.62	0.14	2	2	2	P10	'Upper'	8.90
0.15	0.82	0.03	2	2	3	P10	'Upper'	11.53
0.90	0.10	0.00	1	1	8	P10	'Upper'	15.46
0.00	0.10	0.90	3	3	2	P11	'Upper'	6.33
0.00	0.02	0.98	3	2	5	P11	'Upper'	8.72
0.04	0.33	0.63	3	3	6	P12	'Upper'	5.95
0.12	0.58	0.30	2	2	7	P12	'Upper'	12.11
0.01	0.05	0.94	3	3	2	P13	'Upper'	4.34
0.07	0.55	0.38	2	2	5	P13	'Upper'	9.44
0.75	0.25	0.00	1	1	3	P2	'Upper'	11.01

0.29	0.69	0.02	2	2	11	P3	'Upper'	10.61
0.98	0.02	0.00	1	1	11	P4	'Upper'	12.26
0.00	0.00	1.00	3	3	5	P5	'Upper'	10.62
0.03	0.55	0.42	2	3	6	P5	'Upper'	6.45
0.00	0.21	0.79	3	3	7	P5	'Upper'	11.26
0.64	0.36	0.00	1	1	11	P5	'Upper'	11.48
0.03	0.74	0.24	2	3	5	P6DS5GC9	'Upper'	10.72
0.00	0.17	0.83	3	3	6	P6DS5GC9	'Upper'	5.31
0.09	0.68	0.23	2	3	7	P6DS5GC9	'Upper'	11.12
0.94	0.06	0.00	1	1	11	P6DS5GC9	'Upper'	13.05
0.08	0.66	0.26	2	3	12	P6DS5GC9	'Upper'	8.50
0.04	0.36	0.60	3	3	5	P7GD4	'Upper'	8.15
0.99	0.01	0.00	1	1	8	P7GD4	'Upper'	11.00
0.34	0.55	0.12	2	2	10	P7GD4	'Upper'	6.73
0.00	0.02	0.98	3	2	11	P7GD4	'Upper'	14.48
0.05	0.63	0.32	2	3	5	P8	'Upper'	10.55
0.00	0.01	0.99	3	3	2	P9	'Upper'	4.93
0.15	0.81	0.03	2	2	2	PEQ1	'Lower'	14.22
0.96	0.04	0.00	1	1	3	PEQ1	'Lower'	22.44
0.01	0.29	0.70	3	2	6	PEQ1	'Lower'	7.01
0.96	0.04	0.00	1	1	8	PEQ1	'Lower'	23.74
0.66	0.34	0.00	1	2	10	PEQ1	'Lower'	11.82
0.77	0.23	0.00	1	2	3	PEQ2LOWER	'Lower'	13.70
0.00	0.00	1.00	3	3	7	PEQ2LOWER	'Lower'	4.93
0.06	0.38	0.56	3	3	6	PEQ2UPPER	'Lower'	8.35
0.99	0.01	0.00	1	1	8	PEQ2UPPER	'Lower'	25.02
0.32	0.66	0.02	2	2	10	PEQ2UPPER	'Lower'	9.00
0.75	0.25	0.00	1	1	2	SUBBB1	'Lower'	23.28

0.17	0.81	0.03	2	2	3	SUBBB1	'Lower'	9.52
0.51	0.44	0.05	1	1	6	SUBBB1	'Lower'	10.20
0.11	0.88	0.01	2	2	7	SUBBB1	'Lower'	12.91
0.03	0.96	0.01	2	2	8	SUBBB1	'Lower'	14.89
0.29	0.69	0.01	2	2	10	SUBBB1	'Lower'	16.86
0.53	0.47	0.00	1	1	6	SUBBB2	'Lower'	11.94
0.40	0.60	0.00	2	2	7	SUBBB2	'Lower'	14.90
0.95	0.05	0.00	1	1	2	SUBBB3	'Lower'	22.19
0.80	0.19	0.00	1	1	6	SUBBB3	'Lower'	10.49
0.48	0.51	0.00	2	2	7	SUBBB3	'Lower'	13.04
0.01	0.57	0.42	2	2	8	SUBBB3	'Lower'	6.49
0.15	0.81	0.04	2	2	10	SUBBB3	'Lower'	20.25
0.06	0.93	0.01	2	2	2	SUBBB4	'Lower'	18.60
0.18	0.69	0.13	2	2	3	SUBBB4	'Lower'	8.88
0.28	0.71	0.02	2	1	6	SUBBB4	'Lower'	13.51
0.15	0.85	0.00	2	1	8	SUBBB4	'Lower'	13.67
0.29	0.71	0.00	2	2	10	SUBBB4	'Lower'	21.25
0.38	0.62	0.00	2	2	2	VACA1	'Lower'	17.07
0.05	0.76	0.19	2	2	3	VACA1	'Lower'	9.62
0.03	0.86	0.10	2	2	6	VACA1	'Lower'	13.23
0.43	0.56	0.00	2	1	8	VACA1	'Lower'	16.30
0.64	0.36	0.00	1	1	10	VACA1	'Lower'	19.69
0.83	0.17	0.00	1	2	3	VACA2	'Lower'	11.99
0.01	0.30	0.69	3	2	6	VACA2	'Lower'	8.99
0.98	0.02	0.00	1	1	8	VACA2	'Lower'	25.12
0.22	0.75	0.03	2	2	10	VACA2	'Lower'	10.96
0.67	0.33	0.00	1	1	10	VACA3VV3	'Upper'	12.05
0.77	0.23	0.00	1	1	11	VACA3VV3	'Upper'	20.27

0.97	0.02	0.00	1	1	2	VACA4VV4	'Upper'	6.83
0.76	0.24	0.00	1	2	3	VACA4VV4	'Upper'	16.58
0.99	0.01	0.00	1	1	8	VACA4VV4	'Upper'	17.86
0.00	0.08	0.92	3	3	10	VACA4VV4	'Upper'	11.77
0.07	0.61	0.31	2	3	11	VACA4VV4	'Upper'	12.85
0.44	0.55	0.01	2	1	8	VACA8VV8GD2	'Upper'	13.78
0.11	0.86	0.03	2	2	10	VACA8VV8GD2	'Upper'	12.94
0.19	0.72	0.09	2	2	11	VACA8VV8GD2	'Upper'	8.97
0.01	0.36	0.63	3	2	2	VCM1VV5	'Upper'	7.02
0.00	0.23	0.77	3	3	3	VCM1VV5	'Upper'	12.82
0.94	0.06	0.00	1	1	8	VCM1VV5	'Upper'	13.19
0.10	0.82	0.08	2	2	10	VCM1VV5	'Upper'	6.91
0.76	0.24	0.00	1	2	11	VCM1VV5	'Upper'	16.99
0.81	0.19	0.00	1	1	8	VCM2	'Upper'	11.50
0.00	0.32	0.68	3	3	9	VCM2	'Upper'	12.22
0.00	0.13	0.87	3	3	9	VCM3	'Upper'	12.76
0.73	0.20	0.07	1	2	2	VV2	'Upper'	8.18
0.18	0.72	0.10	2	2	3	VV2	'Upper'	11.82
0.99	0.01	0.00	1	1	8	VV2	'Upper'	11.14
0.57	0.43	0.00	1	1	11	VV7	'Upper'	26.42
0.00	0.01	0.99	3	3	11	VVEDGE	'Upper'	5.90
0.03	0.92	0.05	2	3	11	VVWB	'Upper'	14.20

- Chapter 4

Uploading the data

```
import pandas as pd
import matplotlib.pyplot as plt
import NumPy as np

from matplotlib import rcParams
rcParams['font.family'] = 'sans-serif'
rcParams['font.sans-serif'] = ['Franklin Gothic Book']
rcParams['font.size'] = '18'

df=pd.read_excel('NN_Results_2.xlsx')
df
```

```
     Axis_Prob  Off_Axis_Prob  Margin_Prob  Pred_Class  True_Class  Geobody  \
0     0.200129       0.764206     0.035665           2           2        2
1     0.896818       0.103155     0.000027           1           1        3
2     0.185925       0.479642     0.334434           2           2        6
3     0.001238       0.096551     0.902211           3           3        7
4     0.217810       0.781036     0.001154           2           2        8
..         ...            ...          ...         ...         ...      ...
149   0.176377       0.722527     0.101096           2           2        3
150   0.985616       0.014327     0.000057           1           1        8
151   0.565369       0.434623     0.000009           1           1       11
152   0.000003       0.009516     0.990481           3           3       11
153   0.030788       0.920465     0.048747           2           3       11

    Meas_Sect Complex Set  Thickness
0       CACH1     'Lower'   8.126835
1       CACH1     'Lower'  17.730909
2       CACH1     'Lower'   6.141687
3       CACH1     'Lower'  11.450495
4       CACH1     'Lower'  18.390356
..        ...         ...        ...
149       VV2     'Upper'  11.824580
150       VV2     'Upper'  11.138141
151       VV7     'Upper'  26.417171
152    VVEDGE     'Upper'   5.896554
153      VVWB     'Upper'  14.202635

[154 rows x 9 columns]
```

```
Meas_Sect=df[df['Meas_Sect']=='CACH1']#REPLACE THE NAME OF THE MEASURED SECTION
Meas_Sect=Meas_Sect[['Axis_Prob', 'Off_Axis_Prob', 'Margin_Prob']]
Meas_Sect
```

```
   Axis_Prob  Off_Axis_Prob  Margin_Prob
0   0.200129       0.764206     0.035665
1   0.896818       0.103155     0.000027
2   0.185925       0.479642     0.334434
3   0.001238       0.096551     0.902211
4   0.217810       0.781036     0.001154
```

```
Meas_Sect=Meas_Sect.values
Meas_Sect
```

```
array([[2.00128968e-01, 7.64206174e-01, 3.56648580e-02],
       [8.96817877e-01, 1.03154714e-01, 2.74090607e-05],
       [1.85924709e-01, 4.79641650e-01, 3.34433641e-01],
       [1.23753958e-03, 9.65514118e-02, 9.02211049e-01],
       [2.17809653e-01, 7.81036309e-01, 1.15403762e-03]])
```

Histograms (or Probability Density Functions, PDFs) and Cummulative Distribution Functions, or CDFs

```
def prob(Meas_Sect):
    prob=np.insert(Meas_Sect, 0, 0, 1)
    return prob

def prob_cumsum(Meas_Sect):
    prob_cumsum=Meas_Sect.cumsum(axis=1)
    prob_cumsum=np.insert(prob_cumsum, 0, 0, 1)
    return prob_cumsum

Meas_Sect_probs=prob(Meas_Sect)
Meas_Sect_probs, Meas_Sect_probs.shape
```

```
(array([[0.00000000e+00, 2.00128968e-01, 7.64206174e-01, 3.56648580e-02],
        [0.00000000e+00, 8.96817877e-01, 1.03154714e-01, 2.74090607e-05],
        [0.00000000e+00, 1.85924709e-01, 4.79641650e-01, 3.34433641e-01],
        [0.00000000e+00, 1.23753958e-03, 9.65514118e-02, 9.02211049e-01],
        [0.00000000e+00, 2.17809653e-01, 7.81036309e-01, 1.15403762e-03]]),
 (5, 4))
```

```
Meas_Sect_probs_cum=prob_cumsum(Meas_Sect)
Meas_Sect_probs_cum, Meas_Sect_probs_cum.shape
```

```
(array([[0.        , 0.20012897, 0.96433514, 1.        ],
        [0.        , 0.89681788, 0.99997259, 1.        ],
        [0.        , 0.18592471, 0.66556636, 1.        ],
        [0.        , 0.00123754, 0.09778895, 1.        ],
        [0.        , 0.21780965, 0.99884596, 1.        ]]),
 (5, 4))
```

```
ArchitecturalElementPositionNames=['0', 'Axis (1)', 'Off-axis (2)', 'Margin (3)']
ArchitecturalElementPositionNames
```

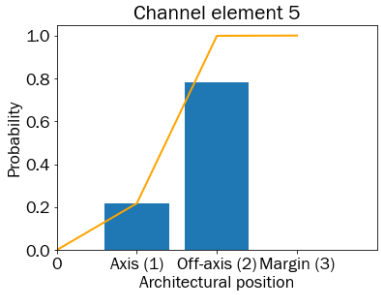
```
['0', 'Axis (1)', 'Off-axis (2)', 'Margin (3)']
```

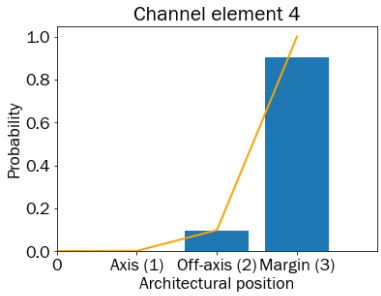
```
ArchitecturalElementPositionNumbers=np.array([0, 1, 2, 3])
print(ArchitecturalElementPositionNumbers.shape, ArchitecturalElementPositionNumbers)
```

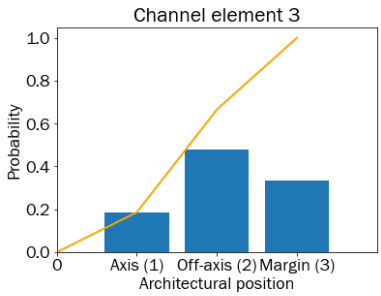
```
(4,) [0 1 2 3]
```

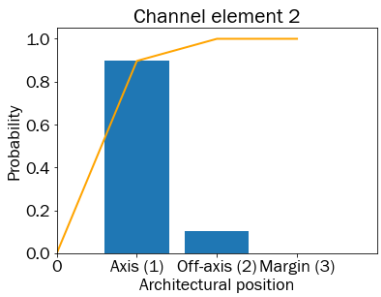
```
def pdfs_and_cdfs(Meas_Sect_probs, Meas_Sect_probs_cum):
    plt.figure(figsize=(6, 22)) #create the figure, then change the size of it
    for c in range(len(Meas_Sect_probs)): #for each sample in the dimension length... THIS IS
USEFUL! TO GO OVER INDEXES OR REPEAT X TIMES A SEQUENCE!!!
        plt.subplot(len(Meas_Sect_probs), 1, len(Meas_Sect_probs)-c) #display the graphs in 4
rows, 3 columns
        plt.bar(ArchitecturalElementPositionNames, Meas_Sect_probs[c]) #plot bars
        plt.plot(Meas_Sect_probs_cum[c], color='orange', linewidth=2) #plot a line above
        plt.title('Channel element '+str(c+1))
        plt.ylabel('Probability') #y label
```

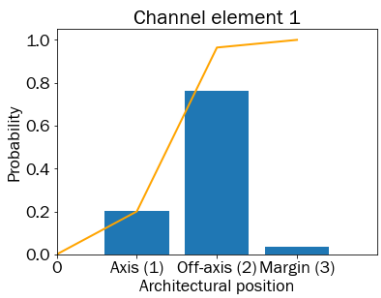
```
        plt.xlabel('Architectural position') #x label
        plt.ylim((0,1.05)) #same dimensions for all the graphs
        plt.xlim((0,4)) #same dimensions for all the graphs
    plt.tight_layout() #improve the layout

pdfs_and_cdfs(Meas_Sect_probs, Meas_Sect_probs_cum)
```



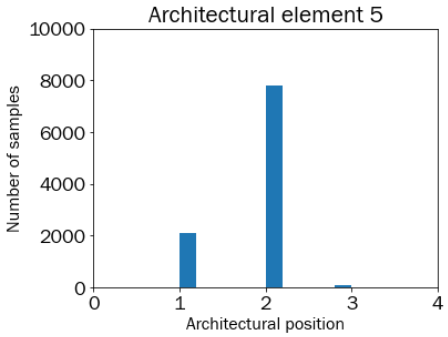


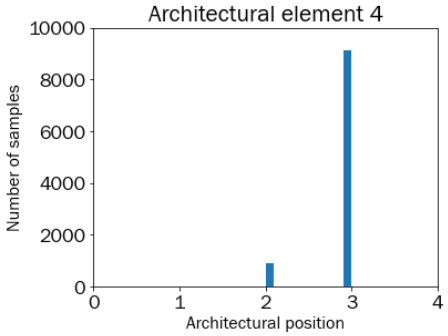


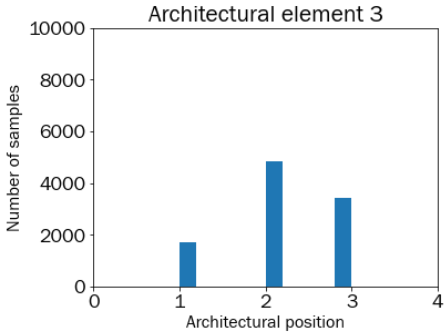


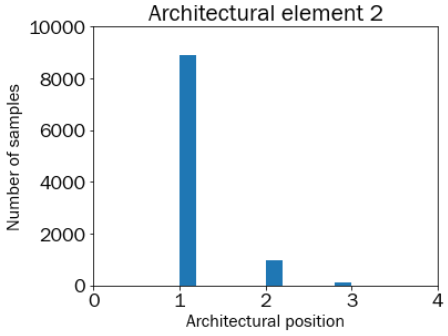


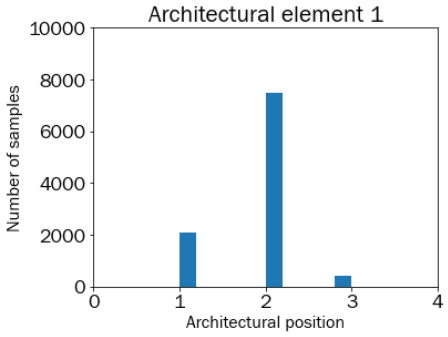
```
import random

n_samples=10000
#EMPTY LIST 1: probability of arch el, (axis, off-axis, margin) per channel element
results=[]
plt.figure(figsize=(6, 22))
for probs in range(len(Meas_Sect_probs_cum)):
    #EMPTY LIST 2: classifications from eCDF (1,2,3)
    mc_sim=[]
    arch_pos=np.array([0, 1, 2, 3])
    a,b,c,d=arch_pos
    w,x,y,z=Meas_Sect_probs_cum[probs]
    plt.subplot(len(Meas_Sect_probs), 1, len(Meas_Sect_probs)-probs)
    #repeat n samples (n_samples)
    for n in range(n_samples):
        value=round(np.random.uniform(0, 1), ndigits=2)
        if w < value <= x:
            arch_el = b
        elif x < value <= y:
            arch_el = c
        else:
            arch_el = d
        # ___OUTPUT 2___
        mc_sim.append(arch_el)
    p_axis=round(mc_sim.count(b)/n_samples, ndigits=2)
    p_offaxis=round(mc_sim.count(c)/n_samples, ndigits=2)
    p_margin=round(mc_sim.count(d)/n_samples, ndigits=2)
    #___OUTPUT 1____
    results.append([p_axis, p_offaxis, p_margin])
    plt.hist(mc_sim)
    plt.title('Architectural element ' + str(probs+1))
    plt.ylabel('Number of samples',fontsize=16) #y label
    plt.xlabel('Architectural position',fontsize=16) #x label
    plt.ylim((0,n_samples)) #same dimensions for all the graphs
    plt.xlim((0,4)) #same dimensions for all the graphs
plt.tight_layout()
results, type(results)

([[0.21, 0.75, 0.04],
  [0.89, 0.1, 0.01],
  [0.17, 0.48, 0.34],
  [0.0, 0.09, 0.91],
  [0.21, 0.78, 0.01]],
 list)
```


Architectural element 5
Number of samples
Architectural position
0
2000
4000
6000
8000
10000
0
1
2
3
4


Architectural element 4
Number of samples
Architectural position
0
2000
4000
6000
8000
10000
0
1
2
3
4


Architectural element 3
Number of samples
Architectural position
0
2000
4000
6000
8000
10000
0
1
2
3
4


Architectural element 2
Number of samples
Architectural position
0
2000
4000
6000
8000
10000
0
1
2
3
4


Architectural element 1
Number of samples
Architectural position
0
2000
4000
6000
8000
10000
0
1
2
3
4

Getting the maximum probabilities

```
arch_pos_codes={1:'axis',
                2:'off-axis',
                3:'margin'}
arch_pos_codes

{1: 'axis', 2: 'off-axis', 3: 'margin'}

max_prob=np.amax(results, axis=1)
max_prob, max_prob.shape, type(max_prob)

(array([0.75, 0.89, 0.48, 0.91, 0.78]), (5,), NumPy.ndarray)

max_final_facies=np.argmax(results,axis=1)+1
max_final_facies, max_final_facies.shape, type(max_final_facies)

(array([2, 1, 2, 3, 2], dtype=int64), (5,), NumPy.ndarray)

ms_final_results_codes_max=[]
for i in range(len(results)):
    codes=arch_pos_codes[max_final_facies[i]]
    ms_final_results_codes_max.append(codes)
ms_final_results_codes_max=np.array(ms_final_results_codes_max)
ms_final_results_codes_max

array(['off-axis', 'axis', 'off-axis', 'margin', 'off-axis'], dtype='<U8')

ms_final_results_max=np.stack((max_final_facies, ms_final_results_codes_max, max_prob), axis=1
)
ms_final_results_max, ms_final_results_max.shape, type(ms_final_results_max)

(array([['2', 'off-axis', '0.75'],
        ['1', 'axis', '0.89'],
        ['2', 'off-axis', '0.48'],
        ['3', 'margin', '0.91'],
        ['2', 'off-axis', '0.78']], dtype='<U32'),
 (5, 3),
 NumPy.ndarray)
```

Getting the minimum probabilities

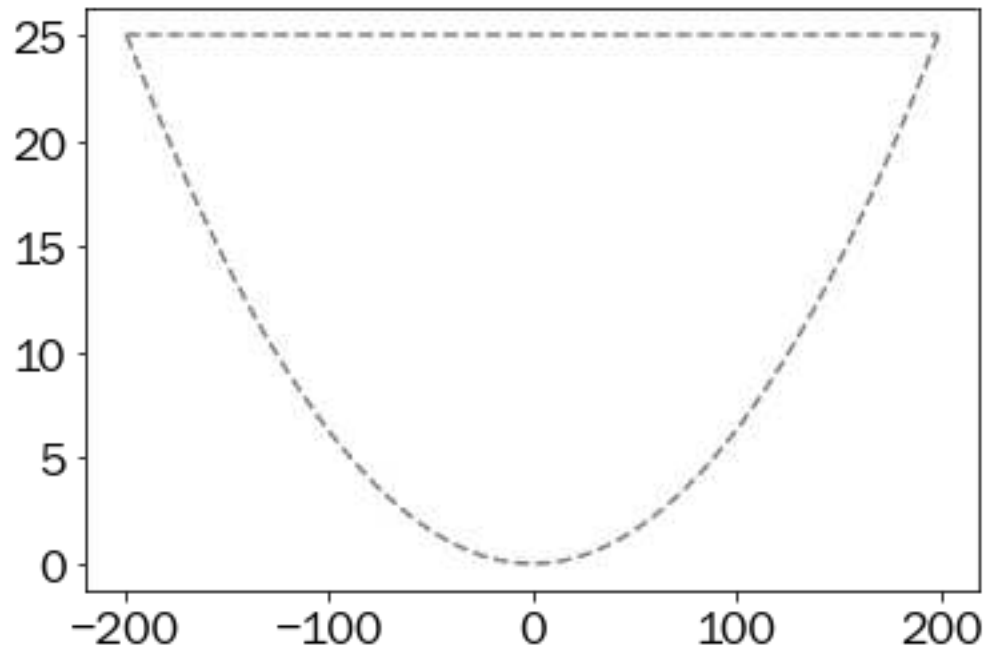
```
min_prob=np.amin(results, axis=1)
min_prob, min_prob.shape, type(min_prob)

(array([0.04, 0.01, 0.17, 0.  , 0.01]), (5,), NumPy.ndarray)

min_final_facies=np.argmin(results,axis=1)+1
min_final_facies, min_final_facies.shape, type(min_final_facies)

(array([3, 3, 1, 1, 3], dtype=int64), (5,), NumPy.ndarray)

ms_final_results_codes_min=[]
for i in range(len(results)):
    codes=arch_pos_codes[min_final_facies[i]]
    ms_final_results_codes_min.append(codes)
ms_final_results_codes_min=np.array(ms_final_results_codes_min)
ms_final_results_codes_min

array(['margin', 'margin', 'axis', 'axis', 'margin'], dtype='<U6')
```

```
ms_final_results_min=np.stack((min_final_facies, ms_final_results_codes_min, min_prob), axis=1
)
ms_final_results_min, ms_final_results_min.shape, type(ms_final_results_min)
```

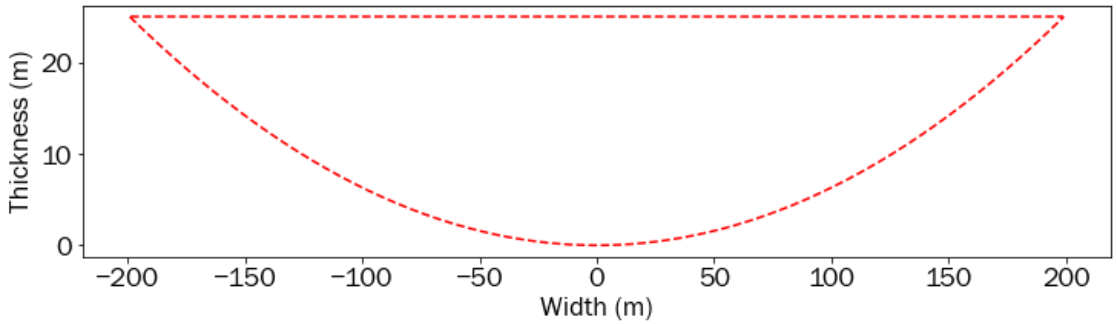
```
(array([['3', 'margin', '0.04'],
        ['3', 'margin', '0.01'],
        ['1', 'axis', '0.17'],
        ['1', 'axis', '0.0'],
        ['3', 'margin', '0.01']], dtype='<U32'),
 (5, 3),
 NumPy.ndarray)
```

Parabola generation

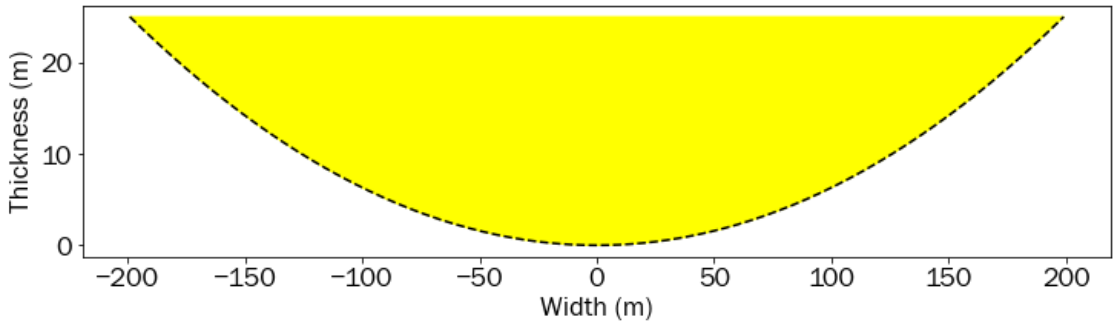
```
def parabola(half_width,height,y_offset=0,x_offset=0):
    xaxis=range(-(half_width-1),half_width)
    xaxis=np.array(xaxis)+x_offset
    yaxis=(height/(half_width-1)**2)*((xaxis-x_offset)**2)+y_offset
    x_top_parabola=[min(xaxis), max(xaxis)]
    y_top_parabola=[max(yaxis), max(yaxis)]
    return plt.plot(xaxis,yaxis,'--',color='gray'), plt.plot(x_top_parabola, y_top_parabola,'-
-',color='gray')

par_test=parabola(half_width=200,height=25,y_offset=0,x_offset=0)
```




```
def parabola_temp(half_width,height,y_offset,x_offset):
    xaxis=range(-(half_width-1),half_width)
    xaxis=np.array(xaxis)+x_offset
    yaxis=(height/(half_width-1)**2)*((xaxis-x_offset)**2)+y_offset
    x_top_parabola=[min(xaxis), max(xaxis)]
    y_top_parabola=[max(yaxis), max(yaxis)]
    return plt.figure(figsize=(12, 3)), plt.plot(xaxis,yaxis,'--',color='red'),plt.plot(x_top_
parabola, y_top_parabola,'--',color='red'),plt.xlabel('Width (m)'), plt.ylabel('Thickness (m)'
)

par_test=parabola_temp(half_width=200,height=25,y_offset=0,x_offset=0)
```



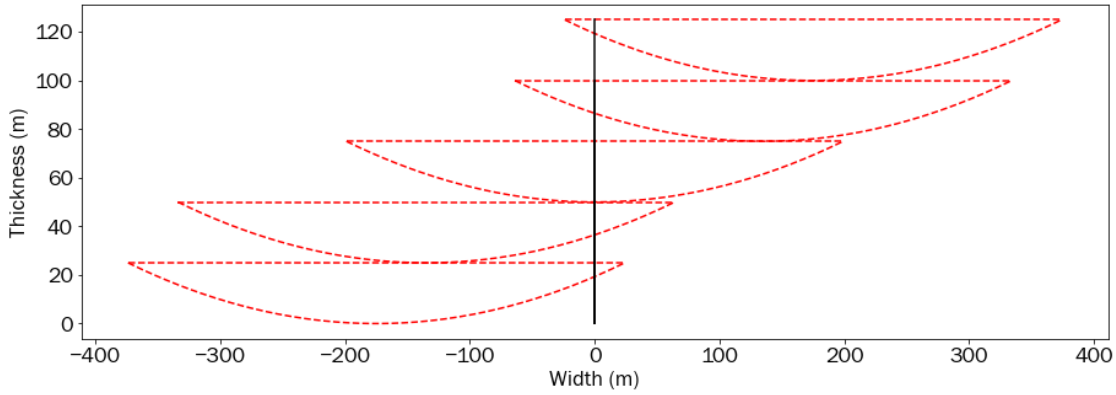

```
def parabola_fill(half_width,height,y_offset=0,x_offset=0):
    xaxis=range(-(half_width-1),half_width)
    xaxis=np.array(xaxis)+x_offset
    yaxis=(height/(half_width-1)**2)*((xaxis-x_offset)**2)+y_offset
    x_top_parabola=[min(xaxis), max(xaxis)]
    y_top_parabola=[max(yaxis), max(yaxis)]
    ms_trace=max(yaxis)
    return plt.figure(figsize=(12, 3)),plt.plot(xaxis,yaxis,'--',color='black'),plt.xlabel('Wi
dth (m)'), plt.ylabel('Thickness (m)'),plt.fill(xaxis,yaxis,'--',color='yellow')

par_fill=parabola_fill(half_width=200,height=25,y_offset=0,x_offset=0)
```



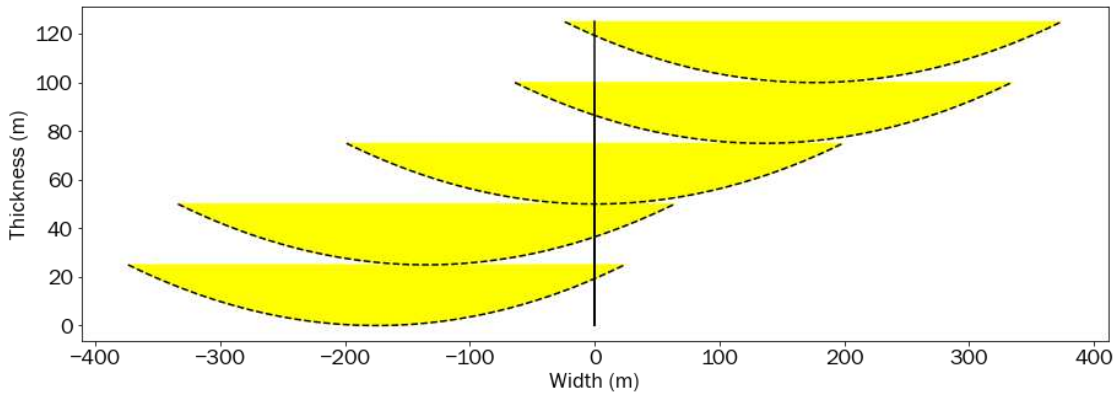
Setting the parameters for the five possible architectural positions

```
width_total=400
perc_to_axis=0
perc_to_offaxis=0.3375
perc_to_margin=0.4375
temp_x_offset=[width_total*-perc_to_margin, width_total*-perc_to_offaxis, width_total*perc_to_
axis, width_total*perc_to_offaxis, width_total*perc_to_margin]
y_offset=25
temp_y_offset=[]
for i in range(len(temp_x_offset)):
    step=y_offset*i
    temp_y_offset.append(step)
print(temp_x_offset, type(temp_x_offset), len(temp_x_offset))
print(temp_y_offset, type(temp_y_offset), len(temp_y_offset))

[-175.0, -135.0, 0, 135.0, 175.0] <class 'list'> 5
[0, 25, 50, 75, 100] <class 'list'> 5

def parabola_temp(half_width,height,y_offset,x_offset): #WORKING...
    xaxis=range(-(half_width-1),half_width)
    xaxis=np.array(xaxis)+x_offset
    yaxis=(height/(half_width-1)**2)*((xaxis-x_offset)**2)+y_offset
```

```
    x_top_parabola=[min(xaxis), max(xaxis)]
    y_top_parabola=[max(yaxis), max(yaxis)]
    ms_trace=max(yaxis)
    return plt.plot(xaxis,yaxis,'--',color='red'),plt.xlabel('Width (m)'), plt.ylabel('Thickne
ss (m)'),plt.plot(x_top_parabola, y_top_parabola,'--',color='red'),plt.plot([0,0], [0,ms_trace
],color='black')

plt.figure(figsize=(15, 5))
for i in range(len(temp_x_offset)):
    parabola_temp(half_width=200,height=25,y_offset=temp_y_offset[i],x_offset=temp_x_offset[i]
)
```




```
def parabola_fill(half_width,height,y_offset=0,x_offset=0):
    xaxis=range(-(half_width-1),half_width)
    xaxis=np.array(xaxis)+x_offset
    yaxis=(height/(half_width-1)**2)*((xaxis-x_offset)**2)+y_offset
    x_top_parabola=[min(xaxis), max(xaxis)]
    y_top_parabola=[max(yaxis), max(yaxis)]
    ms_trace=max(yaxis)
    return plt.plot(xaxis,yaxis,'--',color='black'),plt.xlabel('Width (m)'), plt.ylabel('Thick
ness (m)'),plt.fill(xaxis,yaxis,'--',color='yellow'),plt.plot([0,0], [0,ms_trace],color='black
')

plt.figure(figsize=(15, 5))
for i in range(len(temp_x_offset)):
    parabola_fill(half_width=200,height=25,y_offset=temp_y_offset[i],x_offset=temp_x_offset[i]
)
```



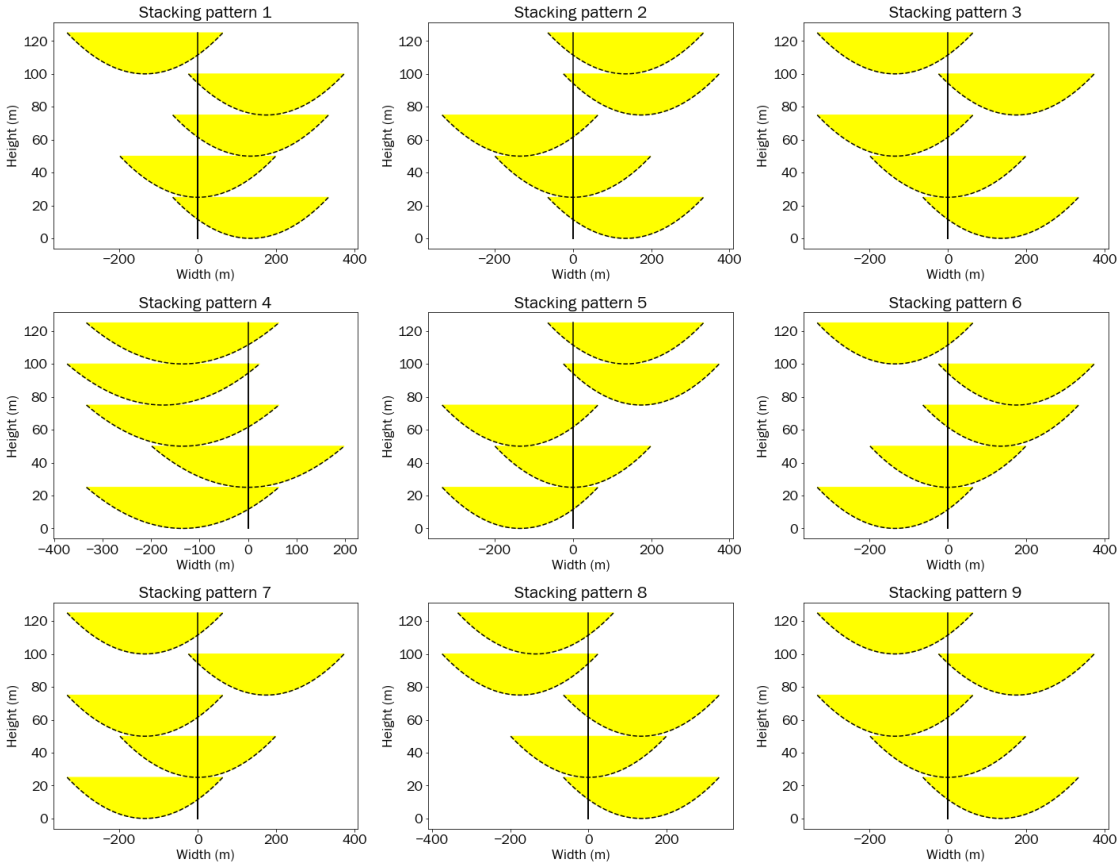
Generation of stacking patterns

```
ms_final_results_max, ms_final_results_max.shape, type(ms_final_results_max)
```

```
(array([['2', 'off-axis', '0.75'],
        ['1', 'axis', '0.89'],
        ['2', 'off-axis', '0.48'],
        ['3', 'margin', '0.91'],
        ['2', 'off-axis', '0.78']], dtype='<U32'),
 (5, 3),
 NumPy.ndarray)
```

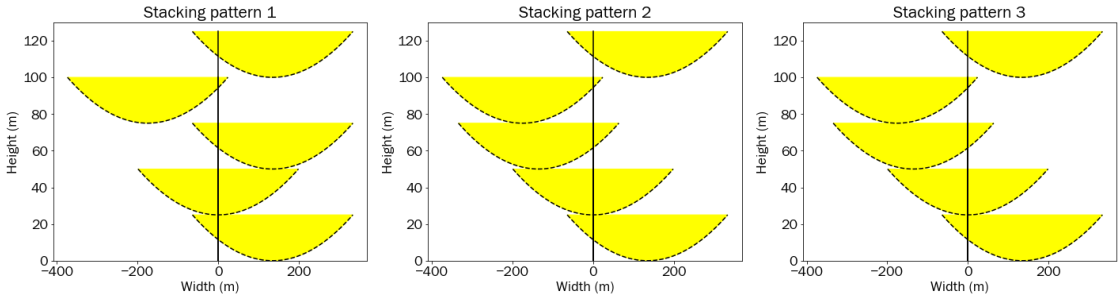
x-offset = 25

```
def stacking_patterns(results, n_stack_patterns, y_offset, width_total=400, perc_to_axis=0, pe
rc_to_offaxis=0.3375, perc_to_margin=0.4375):
    temp_incision=[width_total*-perc_to_margin, width_total*-perc_to_offaxis, width_total*perc
_to_axis, width_total*perc_to_offaxis, width_total*perc_to_margin]
    x_offset_list=[]
    for j in range(n_stack_patterns):
        incision=[]
        for i in range(len(results)):
            rand_margin=np.random.choice([-2,2])
            rand_offaxis=np.random.choice([-1,1])
            rand_axis=0
            if rand_margin==-2 and int(results[i,0])==3:
                inc_step=temp_incision[0]
            else:
                if rand_offaxis==-1 and int(results[i,0])==2:
                    inc_step=temp_incision[1]
                else:
                    if rand_axis==0 and int(results[i,0])==1:
                        inc_step=temp_incision[2]
                    else:
                        if rand_offaxis==1 and int(results[i,0])==2:
                            inc_step=temp_incision[3]
                        else:
                            inc_step=temp_incision[4]
            incision.append(inc_step)
        x_offset_list.append(incision)
    x_offset_list=np.array(x_offset_list)
    y_offset_list=[]
    for k in range(len(incision)):
        agg_step=y_offset*k
        y_offset_list.append(agg_step)
    return x_offset_list, y_offset_list

stack_pat_test=stacking_patterns(ms_final_results_max, 9, 25)
x_offset, y_offset=stack_pat_test

plt.figure(figsize=(20, 20))
for k in range(len(x_offset)):
    hor_disp=x_offset[k]
    plt.subplot(4, 3, k+1)
    for i in range(len(hor_disp)):
        parabola_fill(half_width=200,height=25,y_offset=y_offset[i],x_offset=hor_disp[i])
    plt.title('Stacking pattern '+str(k+1))
    plt.ylabel('Height (m)') #y label
    plt.xlabel('Width (m)') #x label
plt.tight_layout()
```



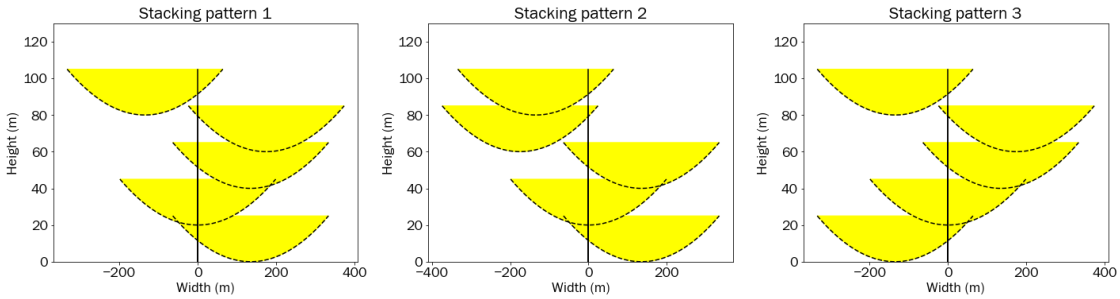
y-offset = 25m

```
stack_pat_25m=stacking_patterns(ms_final_results_max, 3, 25)
x_offset, y_offset=stack_pat_25m

plt.figure(figsize=(20, 20))
for k in range(len(x_offset)):
    hor_disp=x_offset[k]
    plt.subplot(4, 3, k+1)
    for i in range(len(hor_disp)):
        parabola_fill(half_width=200,height=25,y_offset=y_offset[i],x_offset=hor_disp[i])
    plt.ylim([0,130])
    plt.title('Stacking pattern '+str(k+1))
    plt.ylabel('Height (m)') #y label
    plt.xlabel('Width (m)') #x label
plt.tight_layout()
```



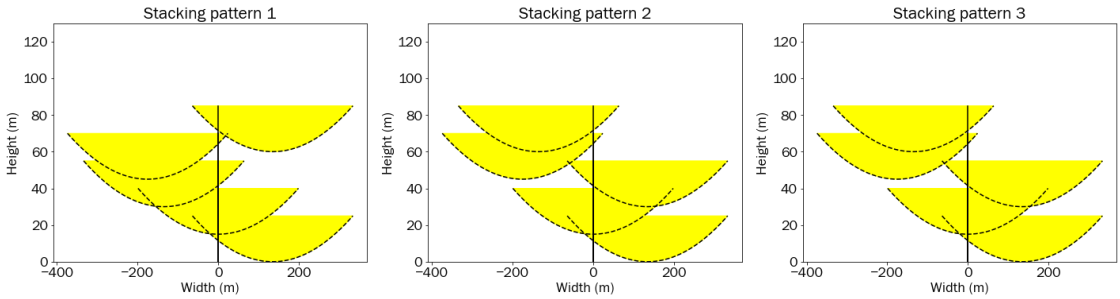
y-offset = 20m

```
stack_pat_20m=stacking_patterns(ms_final_results_max, 3, 20)
x_offset, y_offset=stack_pat_20m

plt.figure(figsize=(20, 20))
for k in range(len(x_offset)):
    hor_disp=x_offset[k]
    plt.subplot(4, 3, k+1)
    for i in range(len(hor_disp)):
        parabola_fill(half_width=200,height=25,y_offset=y_offset[i],x_offset=hor_disp[i])
    plt.ylim([0,130])
    plt.title('Stacking pattern '+str(k+1))
    plt.ylabel('Height (m)') #y label
    plt.xlabel('Width (m)') #x label
plt.tight_layout()
```



y_offset = 15m

```
stack_pat_15m=stacking_patterns(ms_final_results_max, 3, 15)
x_offset, y_offset=stack_pat_15m

plt.figure(figsize=(20, 20))
for k in range(len(x_offset)):
    hor_disp=x_offset[k]
    plt.subplot(4, 3, k+1)
    for i in range(len(hor_disp)):
        parabola_fill(half_width=200,height=25,y_offset=y_offset[i],x_offset=hor_disp[i])
    plt.ylim([0,130])
    plt.title('Stacking pattern '+str(k+1))
    plt.ylabel('Height (m)') #y label
    plt.xlabel('Width (m)') #x label
plt.tight_layout()
```



Assesing multiple realizations of stacking patterns

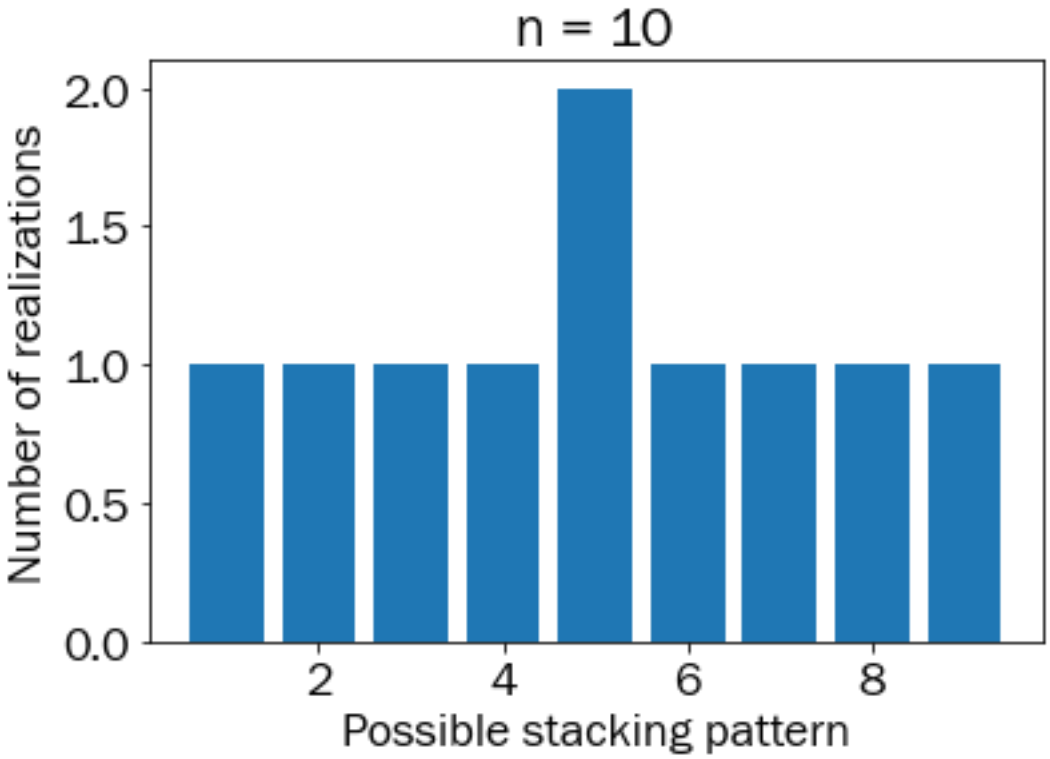
```
def multiple_stack_pat(x_offset):
    possibilities_list=np.unique(x_offset, axis=0).tolist()
```

```
    x_offset_list=x_offset.tolist()
    labels=list(range(1,len(possibilities_list)+1))
    prel_results=[]
    for i in range(len(possibilities_list)):
        counting=x_offset_list.count(possibilities_list[i])
        prel_results.append(counting)
    return plt.bar(labels, prel_results), plt.title('n = '+str(len(x_offset_list))), plt.ylabe
l('Number of realizations'), plt.xlabel('Possible stacking pattern')
```

10 iterations of stacking patterns

```
stack_pat_15m_100=stacking_patterns(ms_final_results_max, 10, 25)
n_incision, aggradation=stack_pat_15m_100
multiple_stack_pat(n_incision)
```

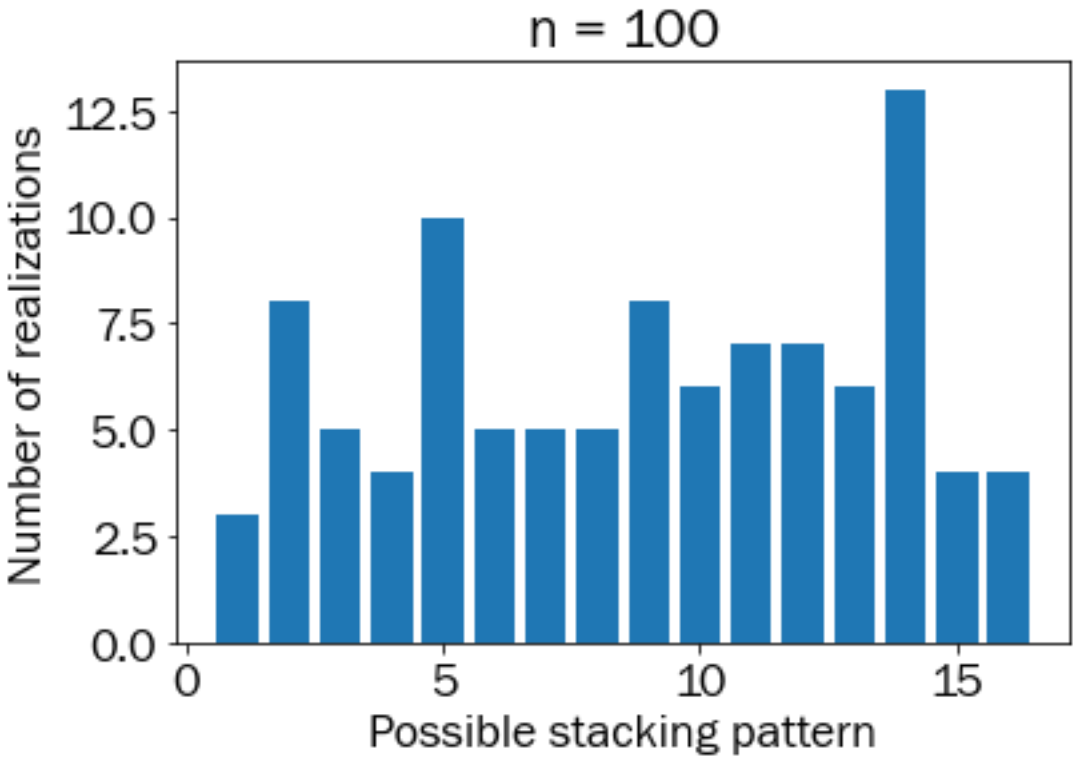
```
(<BarContainer object of 9 artists>,
 Text(0.5, 1.0, 'n = 10'),
 Text(0, 0.5, 'Number of realizations'),
 Text(0.5, 0, 'Possible stacking pattern'))
```



100 iterations of stacking patterns

```
stack_pat_15m_100=stacking_patterns(ms_final_results_max, 100, 25)
n_incision, aggradation=stack_pat_15m_100
multiple_stack_pat(n_incision)
```

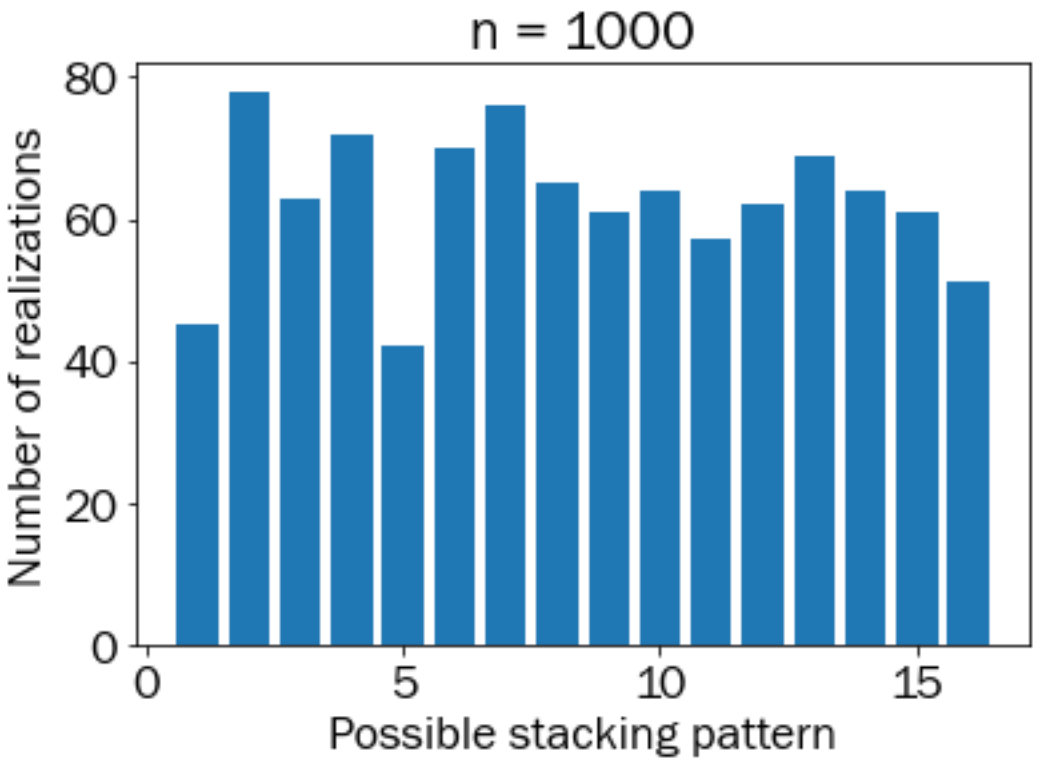
```
(<BarContainer object of 16 artists>,
 Text(0.5, 1.0, 'n = 100'),
 Text(0, 0.5, 'Number of realizations'),
 Text(0.5, 0, 'Possible stacking pattern'))
```



1000 iterations of stacking patterns

```
stack_pat_15m_1000=stacking_patterns(ms_final_results_max, 1000, 25)
n_incision, aggradation=stack_pat_15m_1000
multiple_stack_pat(n_incision)
```

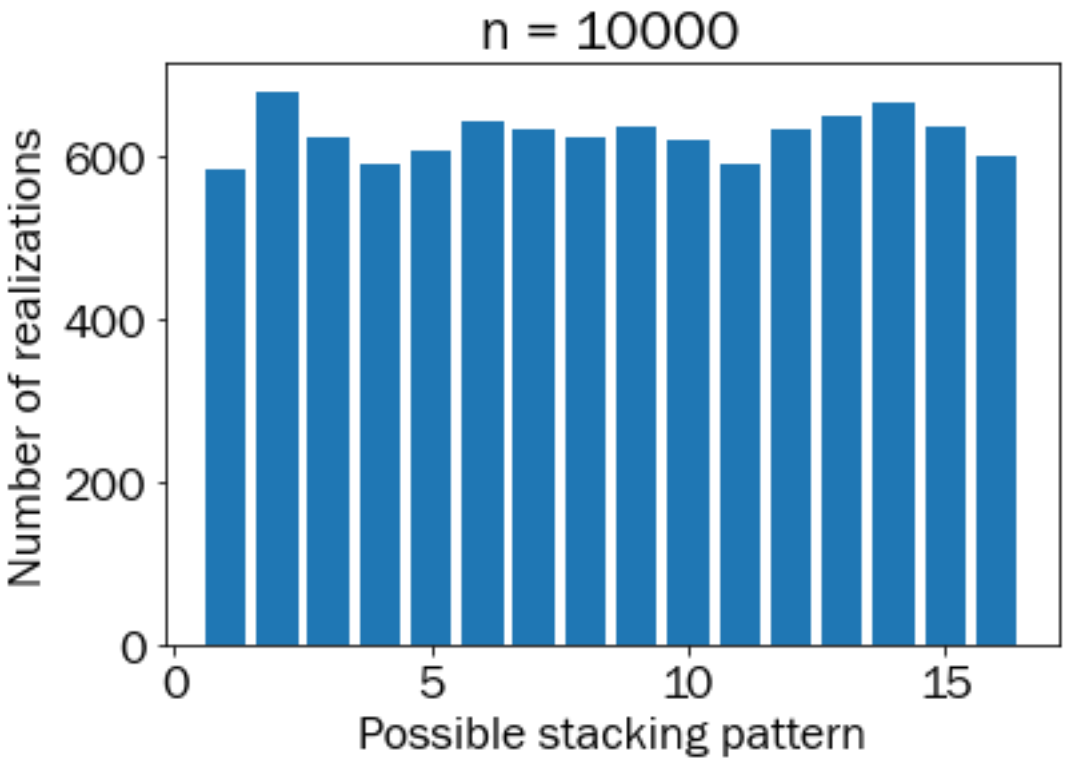
```
(<BarContainer object of 16 artists>,
 Text(0.5, 1.0, 'n = 1000'),
 Text(0, 0.5, 'Number of realizations'),
 Text(0.5, 0, 'Possible stacking pattern'))
```



10000 iterations of stacking patterns

```
stack_pat_15m_10000=stacking_patterns(ms_final_results_max, 10000, 25)
n_incision, aggradation=stack_pat_15m_10000
multiple_stack_pat(n_incision)
```

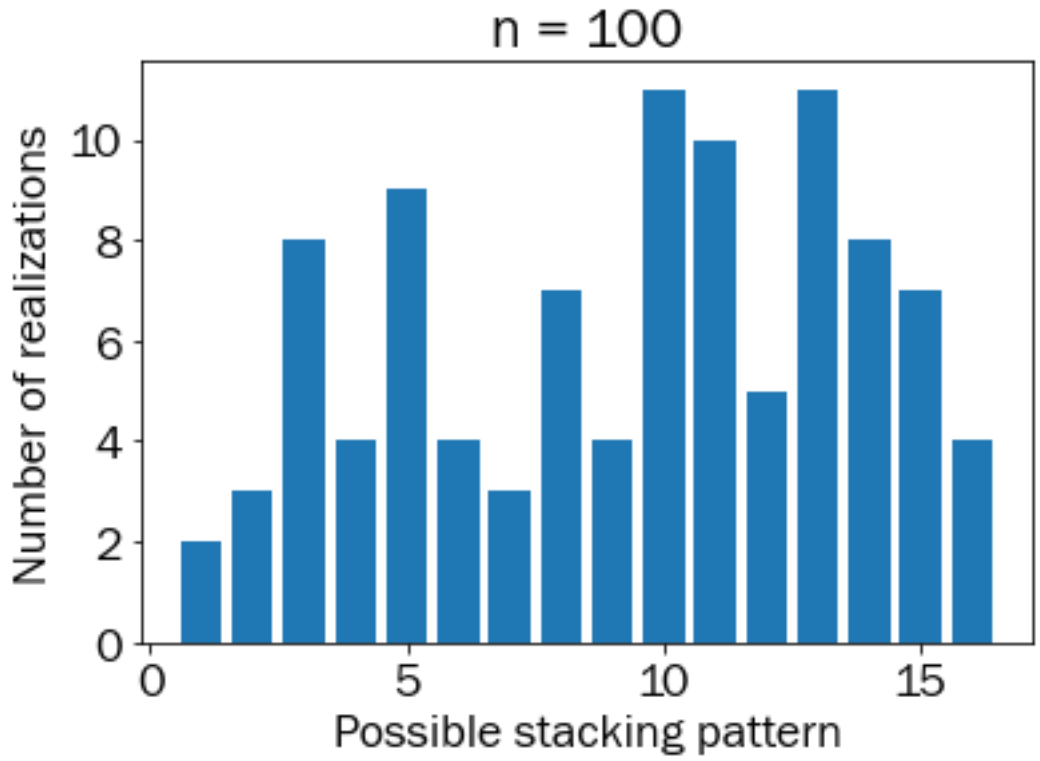
```
(<BarContainer object of 16 artists>,
 Text(0.5, 1.0, 'n = 10000'),
 Text(0, 0.5, 'Number of realizations'),
 Text(0.5, 0, 'Possible stacking pattern'))
```



Generating possible stacking patterns with lateral offset of 25

```
stack_pat_15m_100=stacking_patterns(ms_final_results_max, 100, 25)
n_incision, aggradation=stack_pat_15m_100
multiple_stack_pat(n_incision)
```

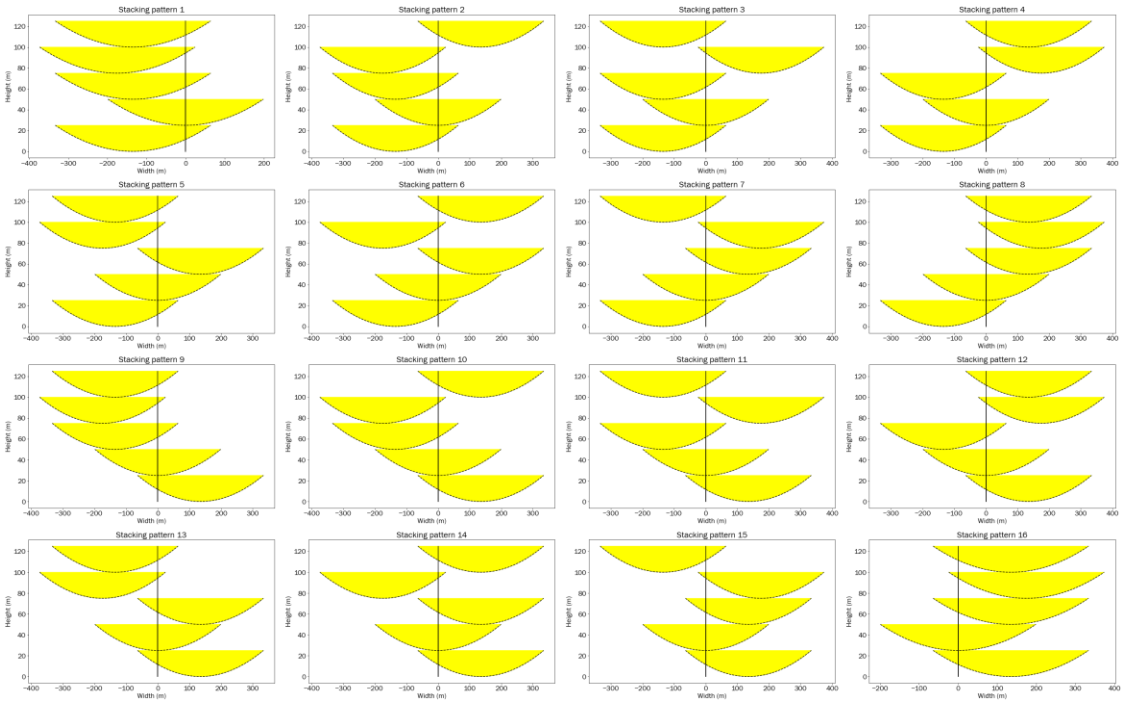
```
(<BarContainer object of 16 artists>,
 Text(0.5, 1.0, 'n = 100'),
 Text(0, 0.5, 'Number of realizations'),
 Text(0.5, 0, 'Possible stacking pattern'))
```




```
unique_possibilities=np.unique(n_incision, axis=0)
unique_possibilities, unique_possibilities.shape
```

```
(array([[-135.,    0., -135., -175., -135.],
        [-135.,    0., -135., -175.,  135.],
        [-135.,    0., -135.,  175., -135.],
        [-135.,    0., -135.,  175.,  135.],
        [-135.,    0.,  135., -175., -135.],
        [-135.,    0.,  135., -175.,  135.],
        [-135.,    0.,  135.,  175., -135.],
        [-135.,    0.,  135.,  175.,  135.],
        [ 135.,    0., -135., -175., -135.],
        [ 135.,    0., -135., -175.,  135.],
        [ 135.,    0., -135.,  175., -135.],
        [ 135.,    0., -135.,  175.,  135.],
        [ 135.,    0.,  135., -175., -135.],
        [ 135.,    0.,  135., -175.,  135.],
        [ 135.,    0.,  135.,  175., -135.],
        [ 135.,    0.,  135.,  175.,  135.]]),
 (16, 5))
```

```
plt.figure(figsize=(40, 25))
for k in range(len(unique_possibilities)):
    hor_disp=unique_possibilities[k]
    plt.subplot(4, 4, k+1)
    for i in range(len(hor_disp)):
        parabola_fill(half_width=200,height=25,y_offset=aggradation[i],x_offset=hor_disp[i])
    plt.title('Stacking pattern '+str(k+1))
    plt.ylabel('Height (m)') #y label
    plt.xlabel('Width (m)') #x label
plt.tight_layout()
```



- Chapter 5

```
import pandas as pd
import matplotlib.pyplot as plt
import NumPy as np

df=pd.read_excel('CSS_DB_v0.2_ElementData_filtered.xlsx')
df
```

```
     NameMS          StudyAreaName  ElementNumber  ComplexNumber  \
0     CACH1  LOWER LAGUNA FIGUEROA              2              2
1     CACH1  LOWER LAGUNA FIGUEROA              3              2
2     CACH1  LOWER LAGUNA FIGUEROA              6              3
3     CACH1  LOWER LAGUNA FIGUEROA              7              3
4     CACH1  LOWER LAGUNA FIGUEROA              8              4
..      ...                    ...            ...            ...
152     VV2  UPPER LAGUNA FIGUEROA              3              1
153     VV2  UPPER LAGUNA FIGUEROA              8              3
154     VV7  UPPER LAGUNA FIGUEROA             11              3
155  VVEDGE  UPPER LAGUNA FIGUEROA             11              3
156    VVWB  UPPER LAGUNA FIGUEROA             11              3

     ComplexSetNumber  ElemWidth  ElementThickness    gs_min    gs_max  \
0                   1        350          8.126835  0.008109  3.663190
1                   1        350         17.730909  0.007651  3.687447
2                   1        350          6.141687  0.007734  0.327069
3                   1        350         11.450495  0.007776  0.329495
4                   1        350         18.390356  0.007551  3.713348
..                ...        ...               ...       ...       ...
152                 2        350         11.824580  0.007664  1.059548
153                 2        350         11.138141  0.138707  0.360099
```

```
154                2        350         26.417171  0.007500  0.852589
155                2        350          5.896554  0.007500  1.787164
156                2        350         14.202635  0.007500  1.771524

       gs_p10  ...  ArchitecturalElementPosition  \
0    0.236430  ...                             2
1    0.224206  ...                             1
2    0.197877  ...                             2
3    0.007816  ...                             3
4    0.183924  ...                             2
..        ...  ...                           ...
152  0.109490  ...                             2
153  0.263874  ...                             1
154  0.193419  ...                             1
155  0.007500  ...                             3
156  0.007552  ...                             3

     MaxPreserved_ElementThickness  MaxPreserved_NTG_gs  MinPreserved_NTG_gs  \
0                        23.284407             1.000000             0.802115
1                        22.442429             0.986979             0.357143
2                        13.505807             0.990228             0.704192
3                        14.895533             0.997481             0.111789
4                        25.782145             1.000000             0.431611
..                             ...                  ...                  ...
152                      16.576063             0.991304             0.565523
153                      21.849939             1.000000             0.302405
154                      26.417171             1.000000             0.252542
155                      26.417171             1.000000             0.252542
156                      26.417171             1.000000             0.252542

     AspectRatio        Net2D      Gross2D     NTG2D  \
0      15.031519  7665.740733  8149.542332  0.940635
1      15.595460  6268.391889  7854.850006  0.798028
2      25.914779  4275.209395  4727.032423  0.904417
3      23.496978  3815.054052  5213.436413  0.731773
4      13.575287  7485.050423  9023.750591  0.829483
..           ...          ...          ...       ...
152    21.114784  5010.105030  5801.622105  0.863570
153    16.018351  6047.025885  7647.478771  0.790721
154    13.248958  7172.708740  9246.010017  0.775763
155    13.248958  7172.708740  9246.010017  0.775763
156    13.248958  7172.708740  9246.010017  0.775763

                                       Architecture  Clinoform
0    Vertically aligned, aggradational slope channe...   Figueroa
1    Vertically aligned, aggradational slope channe...   Figueroa
2    Vertically aligned, aggradational slope channe...   Figueroa
3    Vertically aligned, aggradational slope channe...   Figueroa
4    Vertically aligned, aggradational slope channe...   Figueroa
..                                                 ...        ...
152  Vertically aligned, aggradational slope channe...   Figueroa
153  Vertically aligned, aggradational slope channe...   Figueroa
154  Vertically aligned, aggradational slope channe...   Figueroa
155  Vertically aligned, aggradational slope channe...   Figueroa
156  Vertically aligned, aggradational slope channe...   Figueroa

[157 rows x 42 columns]

data=df.values
data, data.shape, type(data)
```

```
(array([['CACH1', 'LOWER LAGUNA FIGUEROA', 2, ..., 0.940634506940842,
         'Vertically aligned, aggradational slope channel system',
         'Figueroa'],
        ['CACH1', 'LOWER LAGUNA FIGUEROA', 3, ..., 0.798028209805489,
         'Vertically aligned, aggradational slope channel system',
         'Figueroa'],
        ['CACH1', 'LOWER LAGUNA FIGUEROA', 6, ..., 0.904417192935943,
         'Vertically aligned, aggradational slope channel system',
         'Figueroa'],
        ...,
        ['VV7', 'UPPER LAGUNA FIGUEROA', 11, ..., 0.775762596726417,
         'Vertically aligned, aggradational slope channel system',
         'Figueroa'],
        ['VVEDGE', 'UPPER LAGUNA FIGUEROA', 11, ..., 0.775762596726417,
         'Vertically aligned, aggradational slope channel system',
         'Figueroa'],
        ['VVWB', 'UPPER LAGUNA FIGUEROA', 11, ..., 0.775762596726417,
         'Vertically aligned, aggradational slope channel system',
         'Figueroa']], dtype=object),
 (157, 42),
 NumPy.ndarray)

np.unique(data[:,32])

array([1, 2, 3], dtype=object)

measured_sections=np.unique(data[:,0]).tolist()
measured_sections, type(measured_sections), len(measured_sections)

(['CACH1',
  'CACH2',
  'DS1',
  'DYMD1',
  'FIG100',
  'GC1',
  'GC10',
  'GC2',
  'GC3',
  'GC4',
  'GC6',
  'GC7',
  'GC8',
  'GCNOR',
  'GD1',
  'GD3',
  'GD5GC5',
  'GD6KS7',
  'GD7',
  'GD8',
  'GOLDA',
  'GOLDB',
  'GOLDC',
  'GOLDD',
  'KJ1',
  'KS3',
  'MM1',
  'MM101',
  'MM102',
  'MM103',
  'MM2',
  'OP1',
```

```
  'OP2',
  'P10',
  'P11',
  'P12',
  'P13',
  'P2',
  'P3',
  'P4',
  'P5',
  'P6DS5GC9',
  'P7GD4',
  'P8',
  'P9',
  'PEQ1',
  'PEQ2LOWER',
  'PEQ2UPPER',
  'SUBBB1',
  'SUBBB2',
  'SUBBB3',
  'SUBBB4',
  'VACA1',
  'VACA2',
  'VACA3VV3',
  'VACA4VV4',
  'VACA8VV8GD2',
  'VCM1VV5',
  'VCM2',
  'VCM3',
  'VV2',
  'VV7',
  'VVEDGE',
  'VVWB'],
 list,
 64)

measured_sections_arch_ele_pos=[]
for measured_section in measured_sections:
    measured_section_chain=list(df[df['NameMS']==measured_section]['ArchitecturalElementPositi
on'])
    measured_sections_arch_ele_pos.append(measured_section_chain)
measured_sections_arch_ele_pos, len(measured_sections_arch_ele_pos)

([[2, 1, 2, 3, 2],
  [3, 1],
  [3, 3],
  [3],
  [2, 3, 1, 2, 3],
  [3],
  [1],
  [3],
  [1],
  [2],
  [1],
  [1],
  [1, 1],
  [3],
  [1, 2],
  [3, 3],
  [1, 1, 1, 1],
  [3, 1],
  [1],
```

```
  [3],
  [3],
  [2],
  [2],
  [2],
  [1, 2, 2, 1, 3, 2, 3],
  [3],
  [1],
  [3, 2, 1, 1, 3],
  [2, 3, 2, 2, 2, 1],
  [2, 1],
  [3],
  [2],
  [2, 1, 2],
  [2, 2, 1],
  [3, 2],
  [3, 2],
  [3, 2],
  [1],
  [2],
  [1],
  [3, 3, 3, 1],
  [3, 3, 3, 1, 3],
  [3, 1, 2, 2],
  [3],
  [3],
  [2, 1, 2, 1, 2],
  [2, 3],
  [3, 1, 2],
  [1, 2, 1, 2, 2, 2],
  [1, 2, 2],
  [1, 1, 2, 2, 2],
  [2, 2, 1, 1, 2],
  [2, 2, 2, 1, 1],
  [2, 2, 1, 2],
  [1, 1],
  [1, 2, 1, 3, 3],
  [1, 2, 2],
  [2, 3, 1, 2, 2],
  [1, 3],
  [3],
  [2, 2, 1],
  [1],
  [3],
  [3]],
 64)

def trans_prob_pairs(measured_section_chain):
    arch_ele_pos_list_m=np.array(measured_section_chain[:-1])
    arch_ele_pos_list_n=np.array(measured_section_chain[1:])
    pairs=np.stack((arch_ele_pos_list_m, arch_ele_pos_list_n), axis=1).tolist()
    return pairs
#pd.DataFrame(pairs, columns=('From', 'To'))

pairs_total=[]
for measured_section_chain in range(len(measured_sections_arch_ele_pos)):
    pairs_per_ms=trans_prob_pairs(measured_sections_arch_ele_pos[measured_section_chain])
    pairs_total.append(pairs_per_ms)
pairs_total, len(pairs_total)
```

```
([[[2, 1], [1, 2], [2, 3], [3, 2]],
  [[3, 1]],
  [[3, 3]],
  [],
  [[2, 3], [3, 1], [1, 2], [2, 3]],
  [],
  [],
  [],
  [],
  [],
  [],
  [],
  [[1, 1]],
  [],
  [[1, 2]],
  [[3, 3]],
  [[1, 1], [1, 1], [1, 1]],
  [[3, 1]],
  [],
  [],
  [],
  [],
  [],
  [],
  [[1, 2], [2, 2], [2, 1], [1, 3], [3, 2], [2, 3]],
  [],
  [],
  [[3, 2], [2, 1], [1, 1], [1, 3]],
  [[2, 3], [3, 2], [2, 2], [2, 2], [2, 1]],
  [[2, 1]],
  [],
  [],
  [[2, 1], [1, 2]],
  [[2, 2], [2, 1]],
  [[3, 2]],
  [[3, 2]],
  [[3, 2]],
  [],
  [],
  [],
  [[3, 3], [3, 3], [3, 1]],
  [[3, 3], [3, 3], [3, 1], [1, 3]],
  [[3, 1], [1, 2], [2, 2]],
  [],
  [],
  [[2, 1], [1, 2], [2, 1], [1, 2]],
  [[2, 3]],
  [[3, 1], [1, 2]],
  [[1, 2], [2, 1], [1, 2], [2, 2], [2, 2]],
  [[1, 2], [2, 2]],
  [[1, 1], [1, 2], [2, 2], [2, 2]],
  [[2, 2], [2, 1], [1, 1], [1, 2]],
  [[2, 2], [2, 2], [2, 1], [1, 1]],
  [[2, 2], [2, 1], [1, 2]],
  [[1, 1]],
  [[1, 2], [2, 1], [1, 3], [3, 3]],
  [[1, 2], [2, 2]],
  [[2, 3], [3, 1], [1, 2], [2, 2]],
  [[1, 3]],
  [],
  [[2, 2], [2, 1]],
```

```
  [],
  [],
  []],
 64)

final_pairs=[]
for measured_section_pairs in range(len(pairs_total)):
    unpack_ms=pairs_total[measured_section_pairs]
    for pairs_in_ms in range(len(unpack_ms)):
        unpack_pair=unpack_ms[pairs_in_ms]
        final_pairs.append(unpack_pair)
final_pairs_list=final_pairs.copy()
final_pairs=np.array(final_pairs)
final_pairs, final_pairs.shape, len(final_pairs)

(array([[2, 1],
        [1, 2],
        [2, 3],
        [3, 2],
        [3, 1],
        [3, 3],
        [2, 3],
        [3, 1],
        [1, 2],
        [2, 3],
        [1, 1],
        [1, 2],
        [3, 3],
        [1, 1],
        [1, 1],
        [1, 1],
        [3, 1],
        [1, 2],
        [2, 2],
        [2, 1],
        [1, 3],
        [3, 2],
        [2, 3],
        [3, 2],
        [2, 1],
        [1, 1],
        [1, 3],
        [2, 3],
        [3, 2],
        [2, 2],
        [2, 2],
        [2, 1],
        [2, 1],
        [2, 1],
        [1, 2],
        [2, 2],
        [2, 1],
        [3, 2],
        [3, 2],
        [3, 2],
        [3, 3],
        [3, 3],
        [3, 1],
        [3, 3],
        [3, 3],
        [3, 1],
```

```
        [1, 3],
        [3, 1],
        [1, 2],
        [2, 2],
        [2, 1],
        [1, 2],
        [2, 1],
        [1, 2],
        [2, 3],
        [3, 1],
        [1, 2],
        [1, 2],
        [2, 1],
        [1, 2],
        [2, 2],
        [2, 2],
        [1, 2],
        [2, 2],
        [1, 1],
        [1, 2],
        [2, 2],
        [2, 2],
        [2, 2],
        [2, 1],
        [1, 1],
        [1, 2],
        [2, 2],
        [2, 2],
        [2, 1],
        [1, 1],
        [2, 2],
        [2, 1],
        [1, 2],
        [1, 1],
        [1, 2],
        [2, 1],
        [1, 3],
        [3, 3],
        [1, 2],
        [2, 2],
        [2, 3],
        [3, 1],
        [1, 2],
        [2, 2],
        [1, 3],
        [2, 2],
        [2, 1]]),
 (93, 2),
 93)

possibilities=np.unique(final_pairs, axis=0).tolist()
possibilities, type(possibilities), len(possibilities)

([[1, 1], [1, 2], [1, 3], [2, 1], [2, 2], [2, 3], [3, 1], [3, 2], [3, 3]],
 list,
 9)

prel_results=[]
for i in range(len(possibilities)):
    counting=final_pairs_list.count(possibilities[i])
```

```
    prel_results.append(counting)
prel_results, sum(prel_results), len(prel_results)

([9, 18, 5, 15, 17, 7, 8, 7, 7], 93, 9)

prel_results_array=np.array(prel_results).reshape(3,3)
prel_results_df=pd.DataFrame(prel_results_array,columns=['Axis', 'Off-axis', 'Margin'],index=[
'Axis', 'Off-axis', 'Margin'])
prel_results_df

          Axis  Off-axis  Margin
Axis         9        18       5
Off-axis    15        17       7
Margin       8         7       7

final_results=[]
for i in range(len(possibilities)):
    prob_i=prel_results[i]/sum(prel_results)
    final_results.append(prob_i)
final_results=np.array(final_results).reshape(3,3).round(2)
final_results, sum(final_results), len(final_results)

(array([[0.1 , 0.19, 0.05],
        [0.16, 0.18, 0.08],
        [0.09, 0.08, 0.08]]),
 array([0.35, 0.45, 0.21]),
 3)

final_results_df=pd.DataFrame(final_results,columns=['Axis', 'Off-axis', 'Margin'],index=['Axi
s', 'Off-axis', 'Margin'])
final_results_df

          Axis  Off-axis  Margin
Axis      0.10      0.19    0.05
Off-axis  0.16      0.18    0.08
Margin    0.09      0.08    0.08

final_results_norm=[]
for position_prob in final_results:
    for position_prob_ind in position_prob:
        final_results_norm.append(position_prob_ind/sum(position_prob))
final_results_norm

[0.29411764705882354,
 0.5588235294117647,
 0.14705882352941177,
 0.380952380952381,
 0.42857142857142855,
 0.1904761904761905,
 0.36,
 0.32,
 0.32]

final_results_norm_df=pd.DataFrame(np.array(final_results_norm).reshape(3,3).round(2),columns=
['Axis', 'Off-axis', 'Margin'],index=['Axis', 'Off-axis', 'Margin'])
final_results_norm_df

          Axis  Off-axis  Margin
Axis      0.29      0.56    0.15
Off-axis  0.38      0.43    0.19
Margin    0.36      0.32    0.32
```

```
final_results_norm

[0.29411764705882354,
 0.5588235294117647,
 0.14705882352941177,
 0.380952380952381,
 0.42857142857142855,
 0.1904761904761905,
 0.36,
 0.32,
 0.32]
```

Uploading the data

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import math

from matplotlib import rcParams
rcParams['font.family'] = 'sans-serif'
rcParams['font.sans-serif'] = ['Franklin Gothic Book']
rcParams['font.size'] = '18'

df=pd.read_excel('NN_Results_2.xlsx')
df
```

```
     Axis_Prob  Off_Axis_Prob  Margin_Prob  Pred_Class  True_Class  Geobody  \
0     0.200129       0.764206     0.035665           2           2        2
1     0.896818       0.103155     0.000027           1           1        3
2     0.185925       0.479642     0.334434           2           2        6
3     0.001238       0.096551     0.902211           3           3        7
4     0.217810       0.781036     0.001154           2           2        8
..         ...            ...          ...         ...         ...      ...
149   0.176377       0.722527     0.101096           2           2        3
150   0.985616       0.014327     0.000057           1           1        8
151   0.565369       0.434623     0.000009           1           1       11
152   0.000003       0.009516     0.990481           3           3       11
153   0.030788       0.920465     0.048747           2           3       11

    Meas_Sect Complex Set  Thickness
0       CACH1     'Lower'   8.126835
1       CACH1     'Lower'  17.730909
2       CACH1     'Lower'   6.141687
3       CACH1     'Lower'  11.450495
4       CACH1     'Lower'  18.390356
..        ...         ...        ...
149       VV2     'Upper'  11.824580
150       VV2     'Upper'  11.138141
151       VV7     'Upper'  26.417171
152    VVEDGE     'Upper'   5.896554
153      VVWB     'Upper'  14.202635

[154 rows x 9 columns]
```

```
Meas_Sect=df[df['Meas_Sect']=='CACH1'] #REPLACE THE NAME OF THE MEASURED SECTION
Meas_Sect=Meas_Sect[['Axis_Prob', 'Off_Axis_Prob', 'Margin_Prob']]
Meas_Sect
```

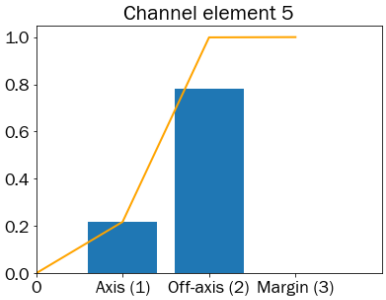
```
   Axis_Prob  Off_Axis_Prob  Margin_Prob
0   0.200129       0.764206     0.035665
```

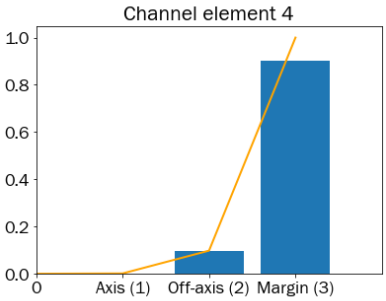
```
1   0.896818       0.103155     0.000027
2   0.185925       0.479642     0.334434
3   0.001238       0.096551     0.902211
4   0.217810       0.781036     0.001154

Meas_Sect=Meas_Sect.values
Meas_Sect

array([[2.00128968e-01, 7.64206174e-01, 3.56648580e-02],
       [8.96817877e-01, 1.03154714e-01, 2.74090607e-05],
       [1.85924709e-01, 4.79641650e-01, 3.34433641e-01],
       [1.23753958e-03, 9.65514118e-02, 9.02211049e-01],
       [2.17809653e-01, 7.81036309e-01, 1.15403762e-03]])
```

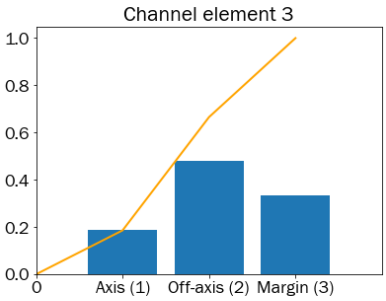
Histograms (or Probability Density Functions, PDFs) and Cummulative Distribution Functions, or CDFs

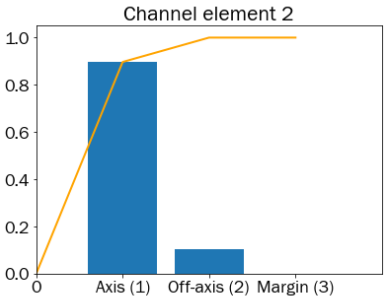
```
def prob(Meas_Sect):
    prob=np.insert(Meas_Sect, 0, 0, 1)
    return prob

def prob_cumsum(Meas_Sect):
    prob_cumsum=Meas_Sect.cumsum(axis=1)
    prob_cumsum=np.insert(prob_cumsum, 0, 0, 1)
    return prob_cumsum

Meas_Sect_probs=prob(Meas_Sect)
Meas_Sect_probs, Meas_Sect_probs.shape

(array([[0.00000000e+00, 2.00128968e-01, 7.64206174e-01, 3.56648580e-02],
        [0.00000000e+00, 8.96817877e-01, 1.03154714e-01, 2.74090607e-05],
        [0.00000000e+00, 1.85924709e-01, 4.79641650e-01, 3.34433641e-01],
        [0.00000000e+00, 1.23753958e-03, 9.65514118e-02, 9.02211049e-01],
        [0.00000000e+00, 2.17809653e-01, 7.81036309e-01, 1.15403762e-03]]),
 (5, 4))

Meas_Sect_probs_cum=prob_cumsum(Meas_Sect)
Meas_Sect_probs_cum, Meas_Sect_probs_cum.shape

(array([[0.        , 0.20012897, 0.96433514, 1.        ],
        [0.        , 0.89681788, 0.99997259, 1.        ],
        [0.        , 0.18592471, 0.66556636, 1.        ],
        [0.        , 0.00123754, 0.09778895, 1.        ],
        [0.        , 0.21780965, 0.99884596, 1.        ]]),
 (5, 4))

ArchitecturalElementPositionNames=['0', 'Axis (1)', 'Off-axis (2)', 'Margin (3)']
ArchitecturalElementPositionNames

['0', 'Axis (1)', 'Off-axis (2)', 'Margin (3)']

ArchitecturalElementPositionNumbers=np.array([0, 1, 2, 3])
print(ArchitecturalElementPositionNumbers.shape, ArchitecturalElementPositionNumbers)

(4,) [0 1 2 3]

def pdfs_and_cdfs(Meas_Sect_probs, Meas_Sect_probs_cum):
    plt.figure(figsize=(6, 22)) #create the figure, then change the size of it
    for c in range(len(Meas_Sect_probs)): #for each sample in the dimension length... THIS IS
USEFUL! TO GO OVER INDEXES OR REPEAT X TIMES A SEQUENCE!!!
        plt.subplot(len(Meas_Sect_probs), 1, len(Meas_Sect_probs)-c) #display the graphs in 4
```

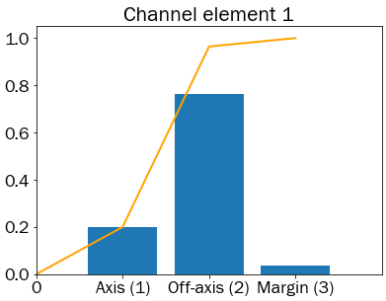
```
rows, 3 columns
        plt.bar(ArchitecturalElementPositionNames, Meas_Sect_probs[c]) #plot bars
        plt.plot(Meas_Sect_probs_cum[c], color='orange', linewidth=2) #plot a line above
        plt.title('Channel element '+str(c+1))
        plt.ylim((0,1.05)) #same dimensions for all the graphs
        plt.xlim((0,4)) #same dimensions for all the graphs
    plt.tight_layout() #improve the layout

pdfs_and_cdfs(Meas_Sect_probs, Meas_Sect_probs_cum)
```












```
import random
```

Luis Carlos: Here's where I made some modifications, in this cell below and the cell after that. With some shifting around, there is a result that produces the most likely random stacking

patterns using the well soft probabilities. The following cell summarized those to see the most likely stacking pattern.

```
def montecarlo(NameMS, NameMS_cumsum, n_samples):
    arch_pos=[0,1,2,3]
    results=[]
    # plt.figure(figsize=(6, 22))
    # repeat n samples (n_samples)
    for n in range(n_samples):
        mc_sim = []
        for probs in range(len(NameMS)):
            #EMPTY LIST 2: classifications from eCDF (1,2,3)

            a,b,c,d=arch_pos
            w,x,y,z=NameMS_cumsum[probs]
            # plt.subplot(len(NameMS), 1, len(NameMS)-probs)

            value=round(np.random.uniform(0, 1), ndigits=2)
            if w < value <= x:
                arch_el = 0
            elif x < value <= y:
                arch_el = np.random.choice([-1,1])
            else:
                arch_el = np.random.choice([-2,2])
            # ___OUTPUT 2___
            mc_sim.append(arch_el)
        # p_axis=round(mc_sim.count(b)/n_samples, ndigits=2)
        # p_offaxis=round(mc_sim.count(c)/n_samples, ndigits=2)
        # p_margin=round(mc_sim.count(d)/n_samples, ndigits=2)
        #___OUTPUT 1____
        # results.append([p_axis, p_offaxis, p_margin])
        results.append(mc_sim)
        # plt.hist(mc_sim)
        # plt.title('Architectural element ' + str(probs+1))
        #plt.ylabel('Number of samples',fontsize=16) #y label
        #plt.xlabel('Architectural position',fontsize=16) #x label
        # plt.ylim((0,n_samples)) #same dimensions for all the graphs
        # plt.xlim((0,4)) #same dimensions for all the graphs
    # plt.tight_layout()
    return results, type(results), pd.DataFrame(results,columns=['Elem1', 'Elem2', 'Elem3','El
em4','Elem5'])

a, b, c =montecarlo(Meas_Sect_probs, Meas_Sect_probs_cum, 100)

# Count Unique STacking Patterns
countrows = c.pivot_table(index = ['Elem1', 'Elem2', 'Elem3', 'Elem4', 'Elem5'], aggfunc = 'si
ze')
countrows = pd.DataFrame(countrows)
countrows.columns = ['count']
countrows = countrows.reindex(countrows.sort_values(by='count', ascending=False).index)
countrows
```

```
                              count
Elem1 Elem2 Elem3 Elem4 Elem5
-1     0     1     2     1         4
              -2    -2     1         4
 1     0     1     2     1         3
-1     0     2     2     1         3
 1     0     1    -2     1         3
...                               ...
```

```
 0     0     1    -2     0         1
-1    -1    -2     2     1         1
 0     0     1    -1    -1         1
                2    -2     0         1
 2     0     2    -1     1         1

[72 rows x 1 columns]
```

```
a, b, c =montecarlo(Meas_Sect_probs, Meas_Sect_probs_cum, 1000000)

# Count Unique STacking Patterns
countrows = c.pivot_table(index = ['Elem1', 'Elem2', 'Elem3', 'Elem4', 'Elem5'], aggfunc = 'si
ze')
countrows = pd.DataFrame(countrows)
countrows.columns = ['count']
countrows = countrows.reindex(countrows.sort_values(by='count', ascending=False).index)
countrows
```

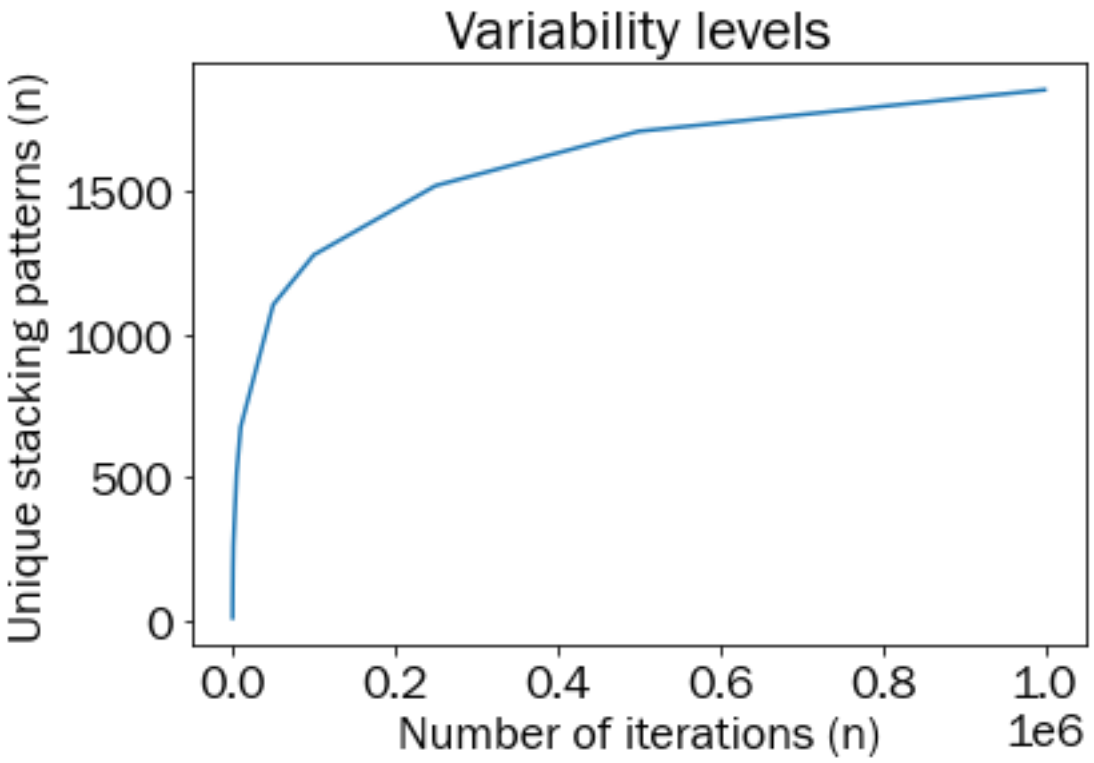
```
                              count
Elem1 Elem2 Elem3 Elem4 Elem5
 1     0    -1     2     1     14584
                   -2    -1     14582
              1     2     1     14565
-1     0     1     2    -1     14547
 1     0    -1     2    -1     14532
...                               ...
 2    -2     0    -2     0         1
-1    -2    -1     2     2         1
              0    -2     2         1
 2    -2    -1    -1    -1         1
              1    -2    -1         1

[1843 rows x 1 columns]
```

```
def var_levels(Meas_Sect_probs, Meas_Sect_probs_cum,experiments_list):
    lenght_rows=[]
    for i in experiments_list:
        a,b,c=montecarlo(Meas_Sect_probs, Meas_Sect_probs_cum, i)
        # Count Unique STacking Patterns
        countrows = c.pivot_table(index = ['Elem1', 'Elem2', 'Elem3', 'Elem4', 'Elem5'], aggfu
nc = 'size')
        countrows = pd.DataFrame(countrows)
        countrows.columns = ['count']
        countrows = countrows.reindex(countrows.sort_values(by='count', ascending=False).index
)
        lenght_rows.append(len(countrows))
    return lenght_rows

experiments_list=[10,100,500,1000,5000,10000,50000,100000,250000,500000,1000000]
lenght_rows=var_levels(Meas_Sect_probs, Meas_Sect_probs_cum,experiments_list)

experiments_list, lenght_rows
```

```
([10, 100, 500, 1000, 5000, 10000, 50000, 100000, 250000, 500000, 1000000],
 [9, 67, 184, 263, 514, 676, 1103, 1277, 1519, 1709, 1854])
```

```
plt.figure()
plt.plot(experiments_list, lenght_rows)
plt.title("Variability levels")
plt.xlabel("Number of iterations (n)")
plt.ylabel("Unique stacking patterns (n)")
```

```
Text(0, 0.5, 'Unique stacking patterns (n)')
```



Getting the 10 most probable results

```
# Count Unique STacking Patterns
countrows = c.pivot_table(index = ['Elem1', 'Elem2', 'Elem3', 'Elem4', 'Elem5'], aggfunc = 'si
ze')
countrows = pd.DataFrame(countrows)
countrows.columns = ['count']
countrows = countrows.reindex(countrows.sort_values(by='count', ascending=False).index)
countrows.head(10)
```

					count
Elem1	Elem2	Elem3	Elem4	Elem5	
1	0	-1	2	1	14584
			-2	-1	14582
		1	2	1	14565
-1	0	1	2	-1	14547
1	0	-1	2	-1	14532
-1	0	-1	2	1	14487
1	0	1	-2	1	14474
-1	0	1	-2	1	14458
1	0	1	2	-1	14456
-1	0	-1	-2	-1	14449

```
# Count Unique STacking Patterns
countrows = c.pivot_table(index = ['Elem1', 'Elem2', 'Elem3', 'Elem4', 'Elem5'], aggfunc = 'si
ze')
countrows = pd.DataFrame(countrows)
countrows.columns = ['count']
countrows = countrows.reindex(countrows.sort_values(by='count', ascending=False).index)
```

```
countrows.head(10)
likelihood=np.array(pd.DataFrame(countrows.to_records()))

most_likely=likelihood[0:10,:-1]
most_likely, most_likely.shape

(array([[ 1,  0, -1,  2,  1],
        [ 1,  0, -1, -2, -1],
        [ 1,  0,  1,  2,  1],
        [-1,  0,  1,  2, -1],
        [ 1,  0, -1,  2, -1],
        [-1,  0, -1,  2,  1],
        [ 1,  0,  1, -2,  1],
        [-1,  0,  1, -2,  1],
        [ 1,  0,  1,  2, -1],
        [-1,  0, -1, -2, -1]], dtype=int64),
 (10, 5))
```

Less likely channel stacking

```
# Count Unique STacking Patterns
countrows = c.pivot_table(index = ['Elem1', 'Elem2', 'Elem3', 'Elem4', 'Elem5'], aggfunc = 'si
ze')
countrows = pd.DataFrame(countrows)
countrows.columns = ['count']
countrows = countrows.reindex(countrows.sort_values(by='count', ascending=False).index)
countrows.tail(10)
```

					count
Elem1	Elem2	Elem3	Elem4	Elem5	
2	-2	1	-1	0	1
-1	-2	-1	2	-2	1
1	-1	2	1	-2	1
2	-2	1	-2	-2	1
		0	1	1	1
			-2	0	1
-1	-2	-1	2	2	1
		0	-2	2	1
2	-2	-1	-1	-1	1
		1	-2	-1	1

```
less_likely=likelihood[-10:,:-1]
less_likely, less_likely.shape

(array([[ 2, -2,  1, -1,  0],
        [-1, -2, -1,  2, -2],
        [ 1, -1,  2,  1, -2],
        [ 2, -2,  1, -2, -2],
        [ 2, -2,  0,  1,  1],
        [ 2, -2,  0, -2,  0],
        [-1, -2, -1,  2,  2],
        [-1, -2,  0, -2,  2],
        [ 2, -2, -1, -1, -1],
        [ 2, -2,  1, -2, -1]], dtype=int64),
 (10, 5))

most_likely[most_likely==-2]=-175
most_likely[most_likely==-1]=-135
most_likely[most_likely==0]=0
most_likely[most_likely==1]=135
```

```
most_likely[most_likely==2]=175
most_likely
```

```
array([[ 135,    0, -135,  175,  135],
       [ 135,    0, -135, -175, -135],
       [ 135,    0,  135,  175,  135],
       [-135,    0,  135,  175, -135],
       [ 135,    0, -135,  175, -135],
       [-135,    0, -135,  175,  135],
       [ 135,    0,  135, -175,  135],
       [-135,    0,  135, -175,  135],
       [ 135,    0,  135,  175, -135],
       [-135,    0, -135, -175, -135]], dtype=int64)
```

Setting the parameters for the five possible architectural positions

```
def parabola_fill(half_width,height,y_offset=0,x_offset=0):
    xaxis=range(-(half_width-1),half_width)
    xaxis=np.array(xaxis)+x_offset
    yaxis=(height/(half_width-1)**2)*((xaxis-x_offset)**2)+y_offset
    x_top_parabola=[min(xaxis), max(xaxis)]
    y_top_parabola=[max(yaxis), max(yaxis)]
    ms_trace=max(yaxis)
    return plt.plot(xaxis,yaxis,'--',color='black'),plt.xlabel('Width (m)'), plt.ylabel('Thick
ness (m)'),plt.fill(xaxis,yaxis,'--',color='yellow'),plt.plot([0,0], [0,ms_trace],color='black
')

thickness=df[df['Meas_Sect']=='CACH1']['Thickness'] #NAME OF THE MEASURED SECTION
thickness=thickness.values
thickness
```

```
array([ 8.12683487, 17.73090935,  6.14168692, 11.45049477, 18.39035606])
```

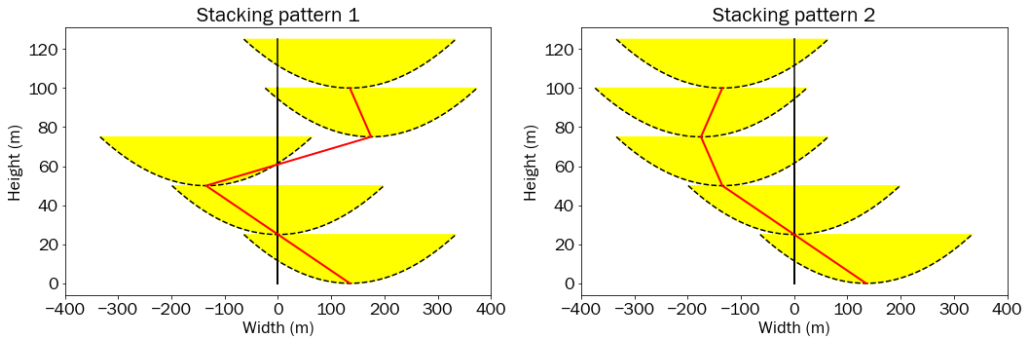
```
def y_offset_to_thickness(results, thicknesses, height=25, width_total=400, perc_to_axis=0, pe
rc_to_offaxis=0.3375, perc_to_margin=0.4375):
    net_erosion=[]
    x_offset=[width_total*perc_to_axis, width_total*perc_to_offaxis, width_total*perc_to_margi
n]
    for c in x_offset:
        value=height/(width_total/2)**2*c**2
        net_erosion.append(value)
    net_y_offset=[]
    arch_pos=results
    for i in range(len(arch_pos)):
        if int(abs(arch_pos[i]))==175:
            value=net_erosion[2]
        elif int(abs(arch_pos[i]))==135:
            value=net_erosion[1]
        else:
            value=net_erosion[0]
        net_y_offset.append(value)
    net_y_offset=np.array(net_y_offset)
    thickness_cumsum=thicknesses.cumsum()
    thickness_cumsum=np.insert(thickness_cumsum[:-1], 0, 0)
    final_y_offset=(thickness_cumsum-net_y_offset).tolist()
    return final_y_offset

y_offset_stack_pattern=[]
for channel_stacking in most_likely:
    ind_y_offset=y_offset_to_thickness(channel_stacking, thickness, height=25, width_total=400
, perc_to_axis=0, perc_to_offaxis=0.3375, perc_to_margin=0.4375)
```

```
    y_offset_stack_pattern.append(ind_y_offset)
y_offset_stack_pattern=np.array(y_offset_stack_pattern)
#y_offset_stack_pattern, y_offset_stack_pattern.shape()
y_offset_stack_pattern, y_offset_stack_pattern.shape
```

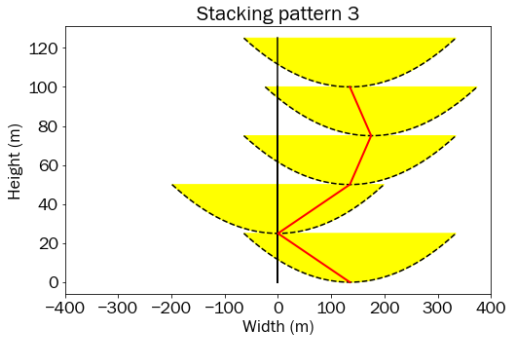
```
(array([[-11.390625  ,   8.12683487,  14.46711922,  12.85880613,
          32.0593009 ],
        [-11.390625  ,   8.12683487,  14.46711922,  12.85880613,
          32.0593009 ],
        [-11.390625  ,   8.12683487,  14.46711922,  12.85880613,
          32.0593009 ],
        [-11.390625  ,   8.12683487,  14.46711922,  12.85880613,
          32.0593009 ],
        [-11.390625  ,   8.12683487,  14.46711922,  12.85880613,
          32.0593009 ],
        [-11.390625  ,   8.12683487,  14.46711922,  12.85880613,
          32.0593009 ],
        [-11.390625  ,   8.12683487,  14.46711922,  12.85880613,
          32.0593009 ],
        [-11.390625  ,   8.12683487,  14.46711922,  12.85880613,
          32.0593009 ],
        [-11.390625  ,   8.12683487,  14.46711922,  12.85880613,
          32.0593009 ],
        [-11.390625  ,   8.12683487,  14.46711922,  12.85880613,
          32.0593009 ]]),
 (10, 5))
```

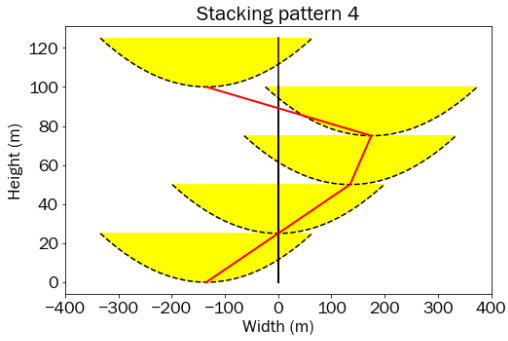
Generating the top 10 most likely channel stacking pattern

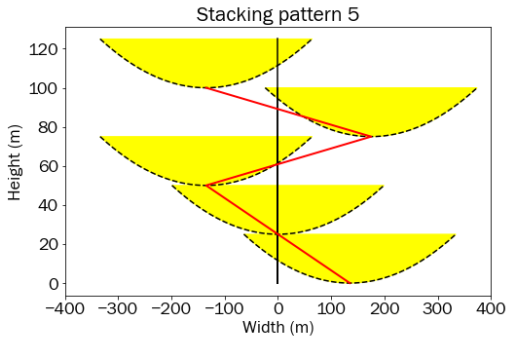
```
y_offset_test=[0,25,50,75,100]

plt.figure(figsize=(15, 25))
for k in range(len(most_likely)):
    hor_disp=most_likely[k]
    plt.subplot(5, 2, k+1)
    for i in range(len(hor_disp)):
        parabola_fill(half_width=200,height=25,y_offset=y_offset_test[i],x_offset=hor_disp[i])
    plt.title('Stacking pattern '+str(k+1))
    plt.ylabel('Height (m)') #y label
    plt.xlabel('Width (m)') #x label
    plt.xlim([-400,400])
    plt.plot(most_likely[k].flatten(), y_offset_test,'red',linewidth=2)
plt.tight_layout()
```

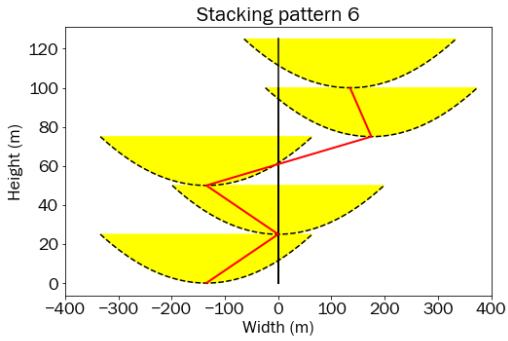

Stacking pattern 1
Height (m)
Width (m)

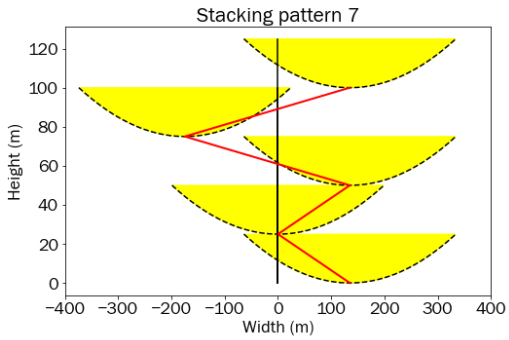
Stacking pattern 2
Height (m)
Width (m)

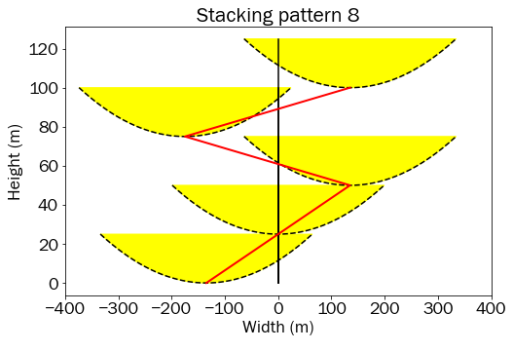

Stacking pattern 3
Height (m)
Width (m)

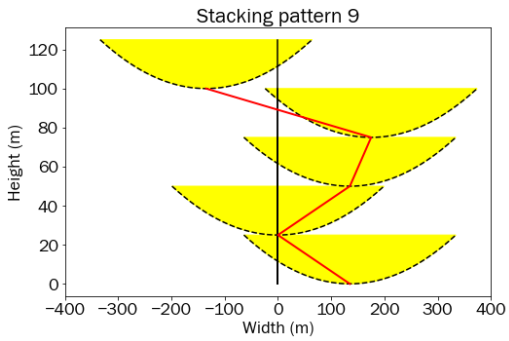

Stacking pattern 4
Height (m)
Width (m)

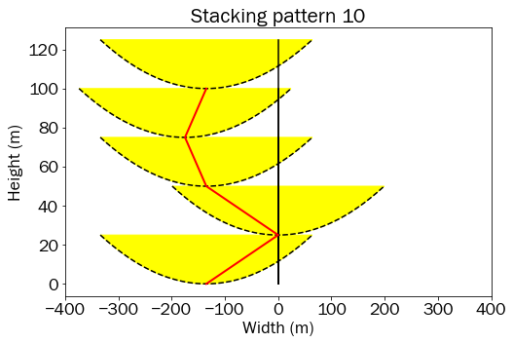

Stacking pattern 5
Height (m)
Width (m)


Stacking pattern 6
Height (m)
Width (m)

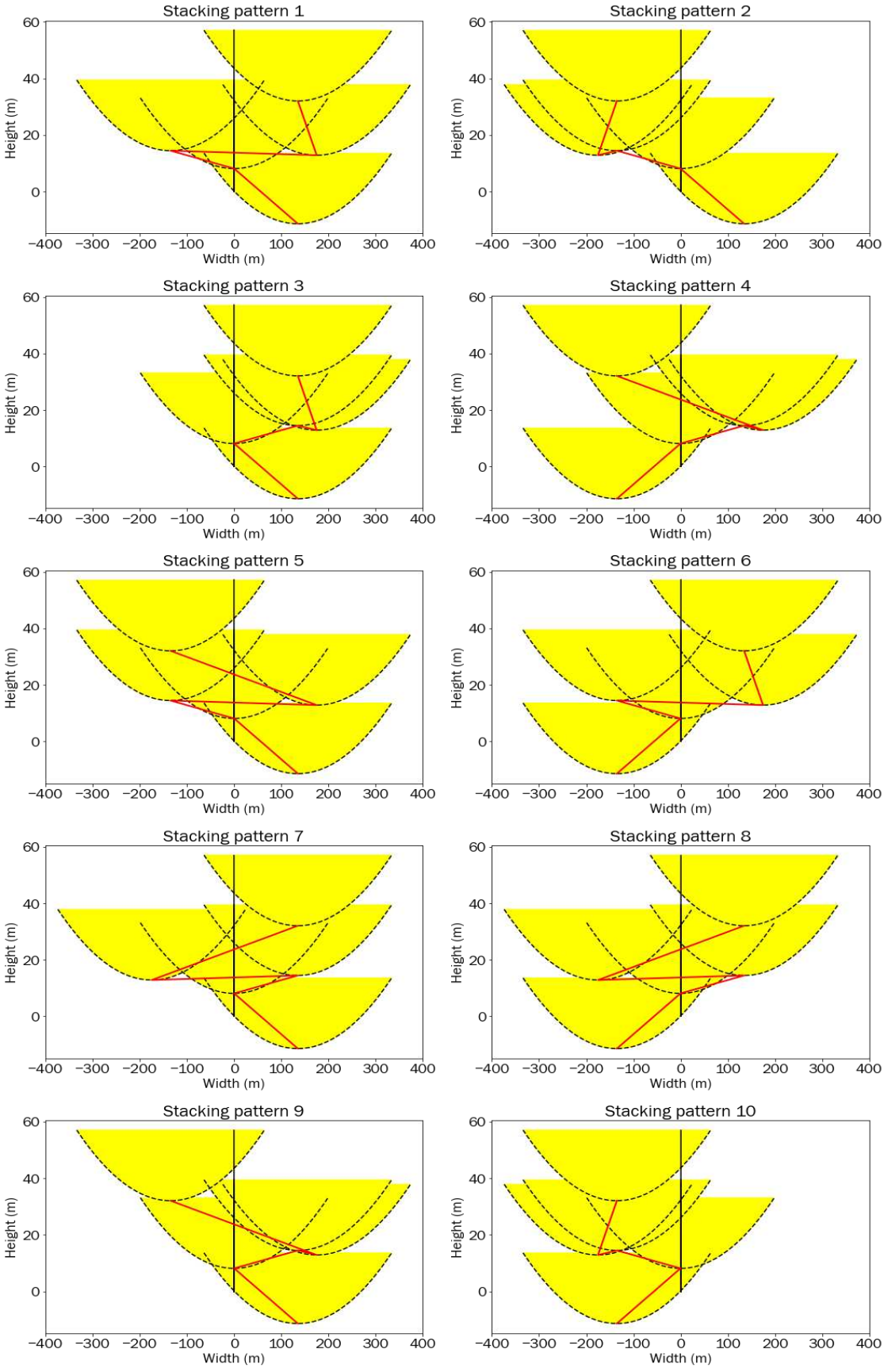

Stacking pattern 7
Height (m)
Width (m)


Stacking pattern 8
Height (m)
Width (m)


Stacking pattern 9
Height (m)
Width (m)


Stacking pattern 10
Height (m)
Width (m)

```
plt.figure(figsize=(15, 25))
for k in range(len(most_likely)):
    hor_disp=most_likely[k]
    ver_disp=y_offset_stack_pattern[k]
    plt.subplot(5, 2, k+1)
    for i in range(len(hor_disp)):
        parabola_fill(half_width=200,height=25,y_offset=ver_disp[i],x_offset=hor_disp[i])
    plt.title('Stacking pattern '+str(k+1))
    plt.ylabel('Height (m)') #y label
    plt.xlabel('Width (m)') #x label
    plt.xlim([-400,400])
    plt.plot(most_likely[k].flatten(), y_offset_stack_pattern[k].flatten(),'red',linewidth=2)
plt.tight_layout()
```

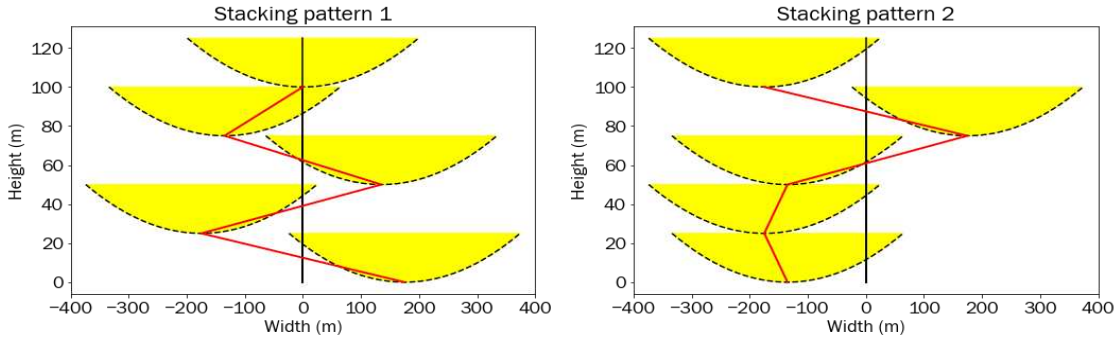

Stacking pattern 1
Stacking pattern 2
Stacking pattern 3
Stacking pattern 4
Stacking pattern 5
Stacking pattern 6
Stacking pattern 7
Stacking pattern 8
Stacking pattern 9
Stacking pattern 10
Height (m)
Width (m)
60
40
20
0
−400
−300
−200
−100
0
100
200
300
400

```
def distances(most_likely, y_offset_stack_pattern):
    total_magnitudes=[]
    for i in range(len(most_likely)):
        lat_offset=most_likely[i]
        ver_offset=y_offset_stack_pattern[i]
        magnitude=[]
        for k in range(len(lat_offset)-1):
            distance=round((((lat_offset[k+1]-lat_offset[k])**2)+((ver_offset[k+1]-ver_offset[
k])**2))**(1/2),2)
            magnitude.append(distance)
        total_magnitudes.append(magnitude)
    return total_magnitudes

distances(most_likely, y_offset_stack_pattern)

[[136.4, 135.15, 310.0, 44.37],
 [136.4, 135.15, 40.03, 44.37],
 [136.4, 135.15, 40.03, 44.37],
 [136.4, 135.15, 40.03, 310.59],
 [136.4, 135.15, 310.0, 310.59],
 [136.4, 135.15, 310.0, 44.37],
 [136.4, 135.15, 310.0, 310.59],
 [136.4, 135.15, 310.0, 310.59],
 [136.4, 135.15, 40.03, 310.59],
 [136.4, 135.15, 40.03, 44.37]]

total_distances=distances(most_likely, y_offset_stack_pattern)
pd.DataFrame(np.array(total_distances).sum(axis=1).reshape(10,1))

        0
0  625.92
1  355.95
2  355.95
3  622.17
4  892.14
5  625.92
6  892.14
7  892.14
8  622.17
9  355.95
```

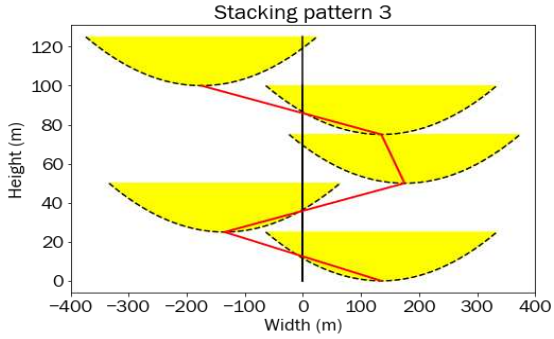
Generating the top less likely channel stacking patterns

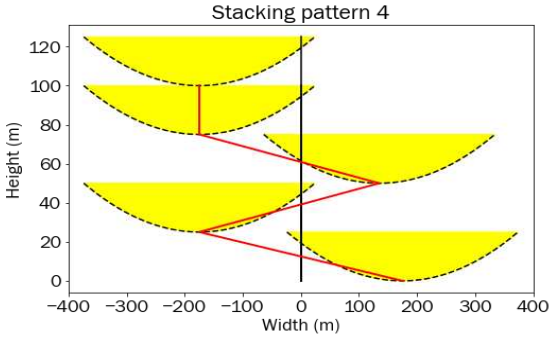
```
less_likely[less_likely==-2]=-175
less_likely[less_likely==-1]=-135
less_likely[less_likely==0]=0
less_likely[less_likely==1]=135
less_likely[less_likely==2]=175
less_likely

array([[ 175, -175,  135, -135,    0],
       [-135, -175, -135,  175, -175],
       [ 135, -135,  175,  135, -175],
       [ 175, -175,  135, -175, -175],
       [ 175, -175,    0,  135,  135],
       [ 175, -175,    0, -175,    0],
       [-135, -175, -135,  175,  175],
       [-135, -175,    0, -175,  175],
       [ 175, -175, -135, -135, -135],
       [ 175, -175,  135, -175, -135]], dtype=int64)
```

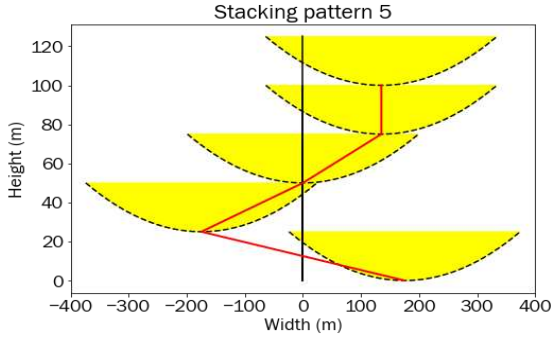
```
y_offset_test=[0,25,50,75,100]

plt.figure(figsize=(15, 25))
for k in range(len(less_likely)):
    hor_disp=less_likely[k]
    plt.subplot(5, 2, k+1)
    for i in range(len(hor_disp)):
        parabola_fill(half_width=200,height=25,y_offset=y_offset_test[i],x_offset=hor_disp[i])
    plt.title('Stacking pattern '+str(k+1))
    plt.ylabel('Height (m)') #y label
    plt.xlabel('Width (m)') #x label
    plt.xlim([-400,400])
    plt.plot(less_likely[k].flatten(), y_offset_test,'red',linewidth=2)
plt.tight_layout()
```

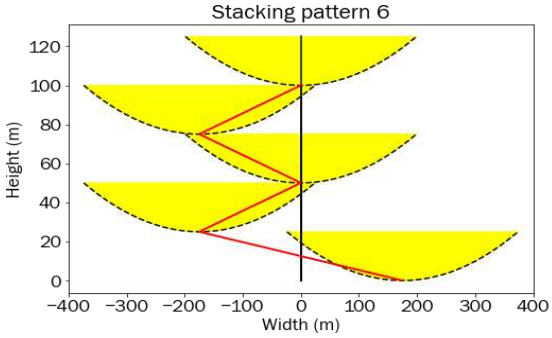

Stacking pattern 1
Height (m)
Width (m)

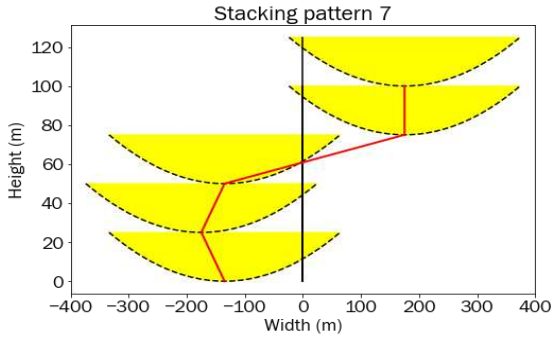
Stacking pattern 2
Height (m)
Width (m)

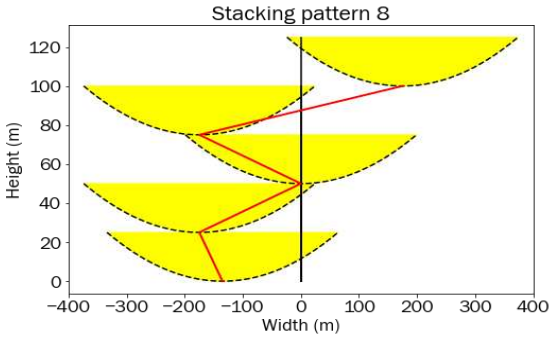

Stacking pattern 3
Height (m)
Width (m)

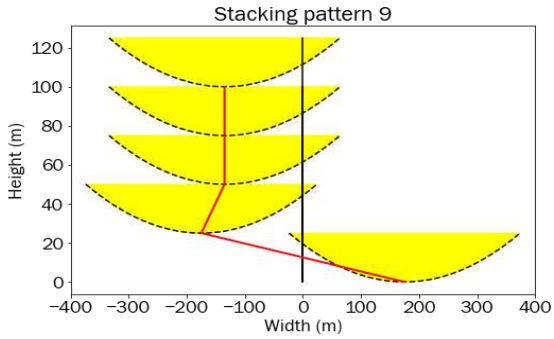

Stacking pattern 4
Height (m)
Width (m)

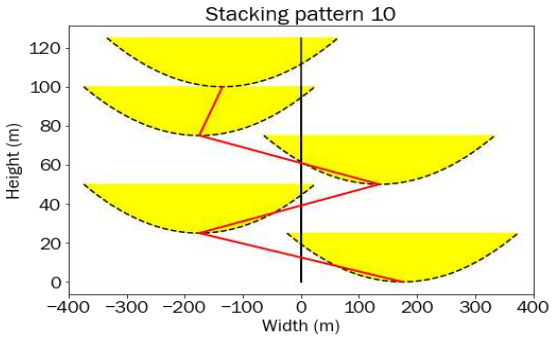

Stacking pattern 5
Height (m)
Width (m)


Stacking pattern 6
Height (m)
Width (m)


Stacking pattern 7
Height (m)
Width (m)


Stacking pattern 8
Height (m)
Width (m)

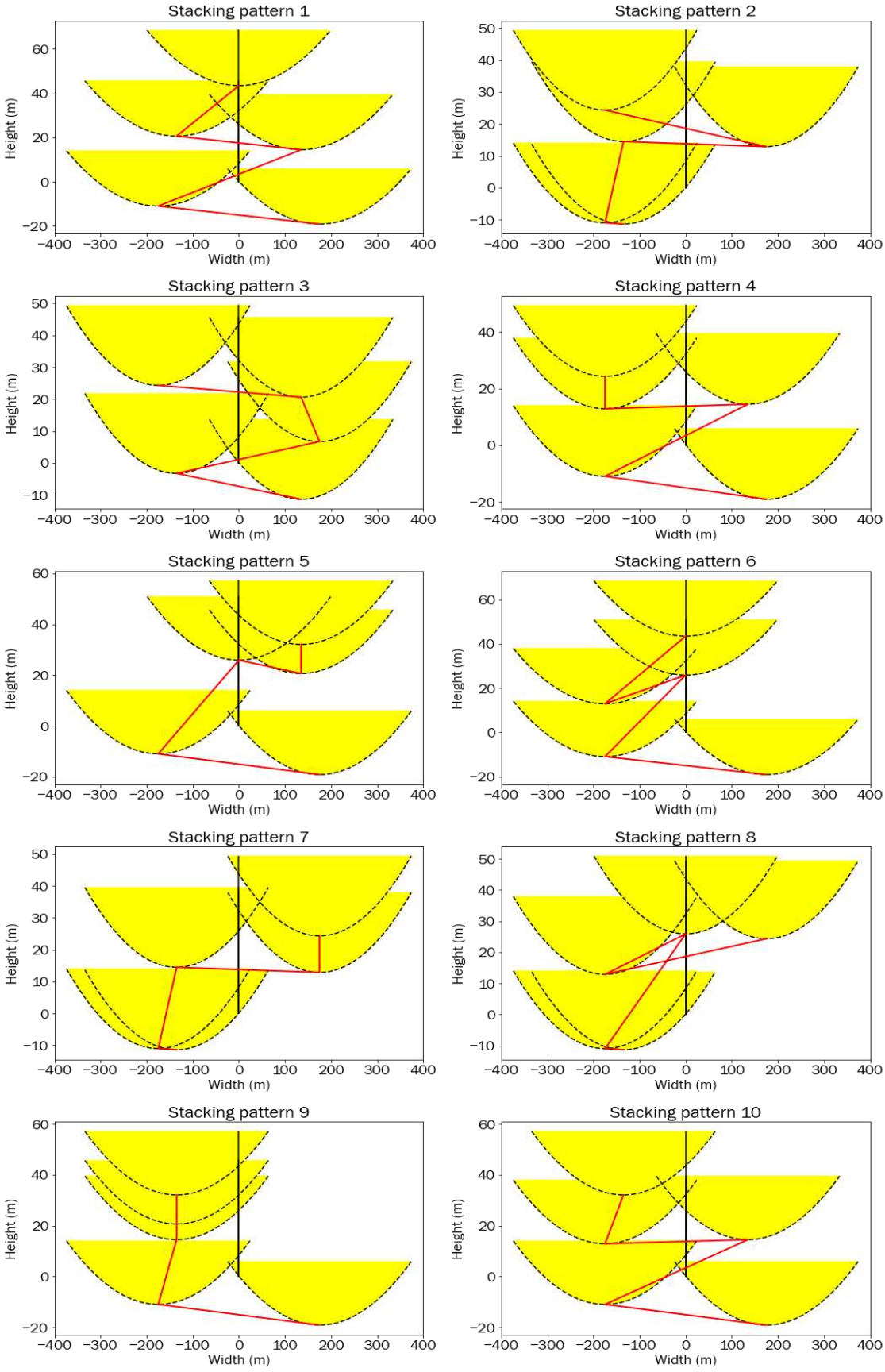

Stacking pattern 9
Height (m)
Width (m)


Stacking pattern 10
Height (m)
Width (m)

```
y_offset_stack_pattern=[]
for channel_stacking in less_likely:
    ind_y_offset=y_offset_to_thickness(channel_stacking, thickness, height=25, width_total=400
, perc_to_axis=0, perc_to_offaxis=0.3375, perc_to_margin=0.4375)
    y_offset_stack_pattern.append(ind_y_offset)
y_offset_stack_pattern=np.array(y_offset_stack_pattern)
#y_offset_stack_pattern, y_offset_stack_pattern.shape()
y_offset_stack_pattern, y_offset_stack_pattern.shape
```

```
(array([[-19.140625  , -11.01379013,  14.46711922,  20.60880613,
          43.4499259 ],
        [-11.390625  , -11.01379013,  14.46711922,  12.85880613,
          24.3093009 ],
        [-11.390625  ,  -3.26379013,   6.71711922,  20.60880613,
          24.3093009 ],
        [-19.140625  , -11.01379013,  14.46711922,  12.85880613,
          24.3093009 ],
        [-19.140625  , -11.01379013,  25.85774422,  20.60880613,
          32.0593009 ],
        [-19.140625  , -11.01379013,  25.85774422,  12.85880613,
          43.4499259 ],
        [-11.390625  , -11.01379013,  14.46711922,  12.85880613,
          24.3093009 ],
        [-11.390625  , -11.01379013,  25.85774422,  12.85880613,
          24.3093009 ],
        [-19.140625  , -11.01379013,  14.46711922,  20.60880613,
          32.0593009 ],
        [-19.140625  , -11.01379013,  14.46711922,  12.85880613,
          32.0593009 ]]),
 (10, 5))
```

```
plt.figure(figsize=(15, 25))
for k in range(len(less_likely)):
    hor_disp=less_likely[k]
    ver_disp=y_offset_stack_pattern[k]
    plt.subplot(5, 2, k+1)
    for i in range(len(hor_disp)):
        parabola_fill(half_width=200,height=25,y_offset=ver_disp[i],x_offset=hor_disp[i])
    plt.title('Stacking pattern '+str(k+1))
    plt.ylabel('Height (m)') #y label
    plt.xlabel('Width (m)') #x label
    plt.xlim([-400,400])
    plt.plot(less_likely[k].flatten(), y_offset_stack_pattern[k].flatten(),'red',linewidth=2)
plt.tight_layout()
```


Stacking pattern 1
Stacking pattern 2
Stacking pattern 3
Stacking pattern 4
Stacking pattern 5
Stacking pattern 6
Stacking pattern 7
Stacking pattern 8
Stacking pattern 9
Stacking pattern 10
Height (m)
Width (m)

```
total_distances=distances(less_likely, y_offset_stack_pattern)
pd.DataFrame(np.array(total_distances).sum(axis=1).reshape(10,1))
```

```
         0
0  1068.13
1   747.62
2   932.64
3   982.59
4   675.48
5   882.06
6   408.88
7   744.51
8   415.11
9  1015.51
```

```
import numpy as np
import random as rm
import matplotlib.pyplot as plt
import pandas as pd

from matplotlib import rcParams
rcParams['font.family'] = 'sans-serif'
rcParams['font.sans-serif'] = ['Franklin Gothic Book']
rcParams['font.size'] = '18'
```

From https://www.datacamp.com/tutorial/markov-chains-python-tutorial

```
# The statespace
states = ["1","2","3"]

# Possible sequences of events
transitionName = [["11","12","13"],["21","22","23"],["31","32","33"]]
```

Transition matrix 1

```
# Probabilities matrix (transition matrix)
transitionMatrix = np.array([[1, 0, 0],
                             [0, 1, 0],
                             [0, 0, 1]])
```

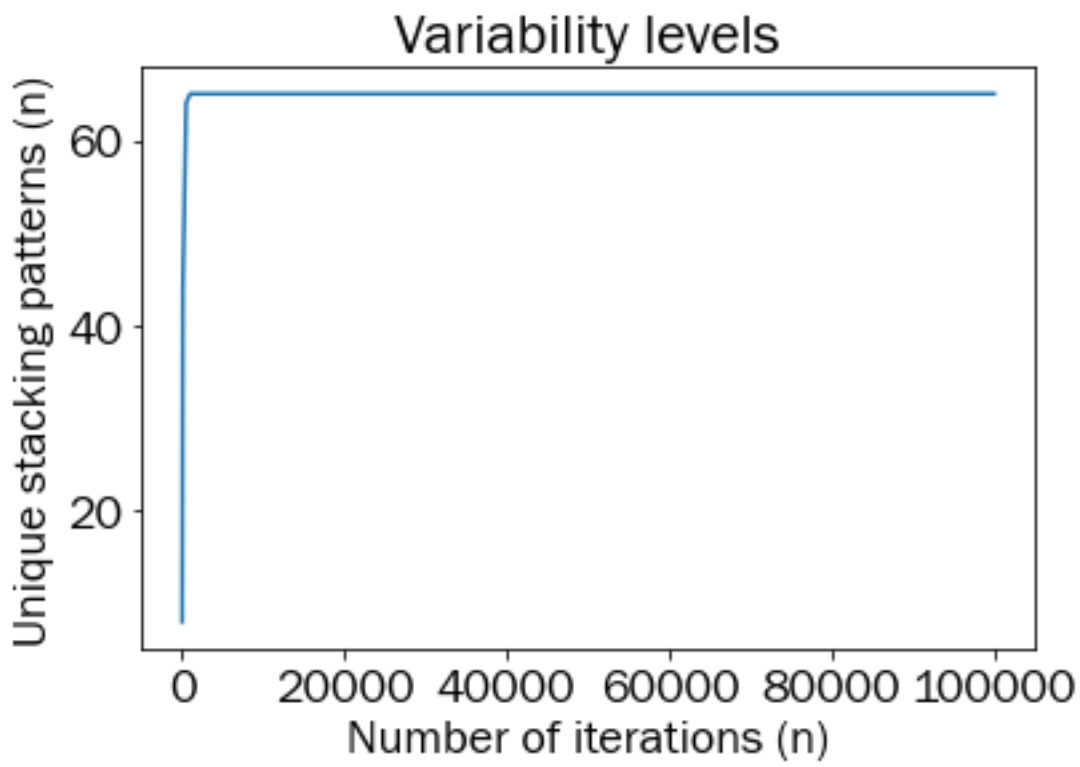
transitionMatrix = np.array([[0.29411764705882354, 0.5588235294117647, 0.14705882352941177], [0.380952380952381, 0.42857142857142855, 0.1904761904761905], [0.36, 0.32, 0.32]])

```
# A function that implements the Markov model to forecast the state/mood.
def forecast(transitionMatrix, future_ch_ele):



    # Probabilities matrix (transition matrix)
#     transitionMatrix = np.array([[0.29411764705882354, 0.5588235294117647, 0.147058823529411
77],
#                          [0.380952380952381, 0.42857142857142855, 0.1904761904761905],
#                          [0.36, 0.32, 0.32]])

#    transitionMatrix = np.array([[1, 0, 0],
#                          [0, 1, 0],
```

```
#                         [0, 0, 1]])
#     transitionMatrix = np.array([[0.3333333333, 0.3333333333, 0.3333333333],
#                         [0.3333333333, 0.3333333333, 0.3333333333],
#                         [0.3333333333, 0.3333333333, 0.3333333333]])
#    transitionMatrix = np.array([[0.1, 0.45, 0.45],
#                        [0.45, 0.1, 0.45],
#                        [0.45, 0.45, 0.1]])
#     if sum(transitionMatrix[0])+sum(transitionMatrix[1])+sum(transitionMatrix[1]) != 3:
#         print("Somewhere, something went wrong. Transition matrix, perhaps?")
#     else: print("All is gonna be okay, you should move on!! ;)")

    # transitionMatrix = [[1,0,0],[0,1,0],[0,0,1]]
    # Choose the starting state
    seed=str(np.random.choice([1,2,3]))
    activityToday = seed
#     print("Start state: " + activityToday)
    # Shall store the sequence of states taken. So, this only has the starting state for now.
    activityList = [activityToday]
    i = 0
    # To calculate the probability of the activityList
    prob = 1
    while i != future_ch_ele:
        if activityToday == "1":
            change = np.random.choice(transitionName[0],replace=True,p=transitionMatrix[0])
            if change == "11":
                prob = prob * transitionMatrix[0,0]
                activityList.append("1")
                pass
            elif change == "12":
                prob = prob * transitionMatrix[0,1]
                activityToday = "2"
                activityList.append("2")
            else:
                prob = prob * transitionMatrix[0,2]
                activityToday = "3"
                activityList.append("3")
        elif activityToday == "2":
            change = np.random.choice(transitionName[1],replace=True,p=transitionMatrix[1])
            if change == "22":
                prob = prob * transitionMatrix[1,1]
                activityList.append("2")
                pass
            elif change == "21":
                prob = prob * transitionMatrix[1,0]
                activityToday = "1"
                activityList.append("1")
            else:
                prob = prob * transitionMatrix[1,2]
                activityToday = "3"
                activityList.append("3")
        elif activityToday == "3":
            change = np.random.choice(transitionName[2],replace=True,p=transitionMatrix[2])
            if change == "33":
                prob = prob * transitionMatrix[2,2]
                activityList.append("3")
                pass
            elif change == "31":
                prob = prob * transitionMatrix[2,0]
                activityToday = "1"
                activityList.append("1")
            else:
```

```
                prob = prob * transitionMatrix[2,1]
                activityToday = "2"
                activityList.append("2")
        i += 1
    #print("Possible states: " + str(activityList))
    #print("End state after "+ str(future_ch_ele) + " future_ch_ele: " + activityToday)
    #print("Probability of the possible sequence of states: " + str(prob))
    return activityList

# Function that forecasts the possible state for the next future_ch_ele
#forecast(4,"2")

def channel_stack_trans_prob(transitionMatrix,channel_stack_number,n_channels):
    channel_stacking=[]
    for n in range(channel_stack_number):
        chan_stack=forecast(transitionMatrix,n_channels-1)
        channel_stacking.append(chan_stack)
    return channel_stacking

def individual_stacking_pattern(result, width_total=400, perc_to_axis=0, perc_to_offaxis=0.337
5, perc_to_margin=0.4375):
    temp_incision=[width_total*-perc_to_margin, width_total*-perc_to_offaxis, width_total*perc
_to_axis, width_total*perc_to_offaxis, width_total*perc_to_margin]
    incision=[]
    for i in range(len(result)):
        rand_margin=np.random.choice([-2,2])
        rand_offaxis=np.random.choice([-1,1])
        rand_axis=0
        if rand_margin==-2 and int(result[i])==3:
            inc_step=temp_incision[0]
        else:
            if rand_offaxis==-1 and int(result[i])==2:
                inc_step=temp_incision[1]
            else:
                if rand_axis==0 and int(result[i])==1:
                    inc_step=temp_incision[2]
                else:
                    if rand_offaxis==1 and int(result[i])==2:
                        inc_step=temp_incision[3]
                    else:
                        inc_step=temp_incision[4]
        incision.append(inc_step)
    #y_offset_list=[]
    #for k in range(len(incision)):
        #agg_step=y_offset*k
        #y_offset_list.append(agg_step)
    return incision#, y_offset_list

def stacking_pattern_trans_prob(channel_stack_list):
    stacking_patterns=[]
    for i in range(len(channel_stack_list)):
        result_step=individual_stacking_pattern(channel_stack_list[i], width_total=400, perc_t
o_axis=0, perc_to_offaxis=0.3375, perc_to_margin=0.4375)
        stacking_patterns.append(result_step)
    stacking_patterns=np.array(stacking_patterns)
    return pd.DataFrame(stacking_patterns,columns=['Elem1', 'Elem2', 'Elem3','Elem4','Elem5'])

def var_levels(transitionMatrix,experiments_list,n_channels):
    lenght_rows=[]
    for i in experiments_list:
        channel_stack=channel_stack_trans_prob(transitionMatrix,i,n_channels)
```

```
        stack_pattern=stacking_pattern_trans_prob(channel_stack)
        #a,b,c=montecarlo(Meas_Sect_probs, Meas_Sect_probs_cum, i)
        # Count Unique STacking Patterns
        countrows = stack_pattern.pivot_table(index = ['Elem1', 'Elem2', 'Elem3', 'Elem4', 'El
em5'], aggfunc = 'size')
        countrows = pd.DataFrame(countrows)
        countrows.columns = ['count']
        countrows = countrows.reindex(countrows.sort_values(by='count', ascending=False).index
)
        lenght_rows.append(len(countrows))
    return lenght_rows

experiments_list=[10,100,500,1000,5000,10000,50000,100000]
lenght_rows=var_levels(transitionMatrix,experiments_list, 5)
experiments_list, lenght_rows

([10, 100, 500, 1000, 5000, 10000, 50000, 100000],
 [8, 44, 64, 65, 65, 65, 65, 65])

plt.figure()
plt.plot(experiments_list, lenght_rows)
plt.title("Variability levels")
plt.xlabel("Number of iterations (n)")
plt.ylabel("Unique stacking patterns (n)")

Text(0, 0.5, 'Unique stacking patterns (n)')
```



Getting the 10 most probable results

```
trans_prob_ch_stack=channel_stack_trans_prob(transitionMatrix,100000,5)
trans_prob_stack_pat=stacking_pattern_trans_prob(trans_prob_ch_stack)
trans_prob_stack_pat
```

```
       Elem1  Elem2  Elem3  Elem4  Elem5
0     -175.0 -175.0  175.0 -175.0 -175.0
1     -135.0 -135.0 -135.0 -135.0  135.0
2     -175.0 -175.0  175.0  175.0  175.0
3      135.0 -135.0 -135.0  135.0 -135.0
4      135.0  135.0  135.0  135.0 -135.0
...      ...    ...    ...    ...    ...
99995  175.0  175.0 -175.0 -175.0 -175.0
99996  175.0 -175.0  175.0 -175.0 -175.0
99997 -135.0 -135.0 -135.0  135.0 -135.0
99998 -175.0  175.0 -175.0  175.0 -175.0
99999 -135.0 -135.0  135.0 -135.0 -135.0

[100000 rows x 5 columns]
```

```
# Count Unique STacking Patterns
countrows = trans_prob_stack_pat.pivot_table(index = ['Elem1', 'Elem2', 'Elem3', 'Elem4', 'Ele
m5'], aggfunc = 'size')
countrows = pd.DataFrame(countrows)
countrows.columns = ['count']
countrows = countrows.reindex(countrows.sort_values(by='count', ascending=False).index)
countrows.head(10)
#likelihood=np.array(pd.DataFrame(countrows.to_records()))
```

```
                                  count
Elem1  Elem2  Elem3  Elem4  Elem5
 0.0    0.0    0.0    0.0    0.0    33338
-135.0 -135.0 -135.0  135.0 -135.0   1128
-175.0 -175.0 -175.0  175.0 -175.0   1122
-135.0 -135.0  135.0  135.0 -135.0   1115
 175.0  175.0  175.0 -175.0  175.0   1094
-175.0  175.0  175.0  175.0 -175.0   1091
                              175.0   1084
 135.0 -135.0  135.0  135.0 -135.0   1084
-135.0 -135.0 -135.0 -135.0 -135.0   1084
 135.0 -135.0 -135.0  135.0 -135.0   1081
```

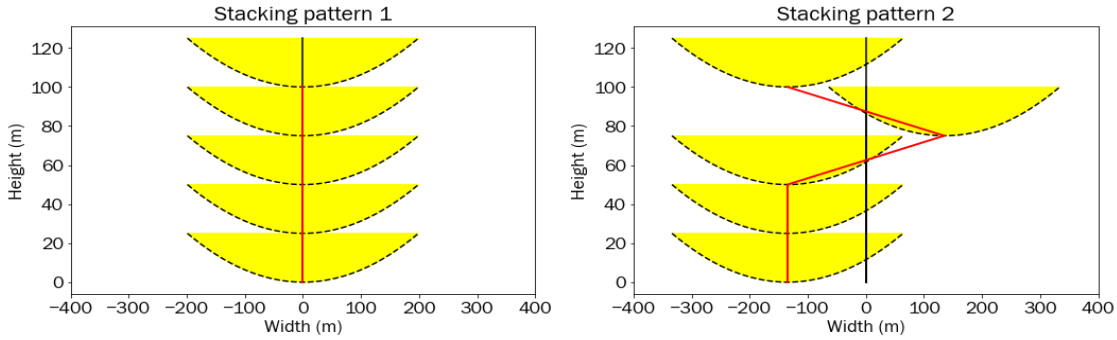
```
likelihood=np.array(pd.DataFrame(countrows.to_records()))
most_likely=likelihood[0:10,:-1]
most_likely, most_likely.shape
```

```
(array([[   0.,    0.,    0.,    0.,    0.],
        [-135., -135., -135.,  135., -135.],
        [-175., -175., -175.,  175., -175.],
        [-135., -135.,  135.,  135., -135.],
        [ 175.,  175.,  175., -175.,  175.],
        [-175.,  175.,  175.,  175., -175.],
        [-175.,  175.,  175.,  175.,  175.],
        [ 135., -135.,  135.,  135., -135.],
        [-135., -135., -135., -135., -135.],
        [ 135., -135., -135.,  135., -135.]]),
 (10, 5))
```

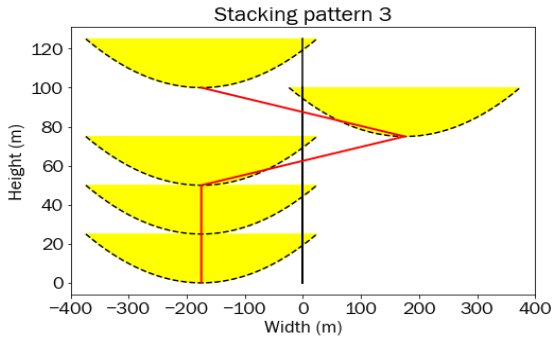
Generating the top 10 most likely channel stacking pattern

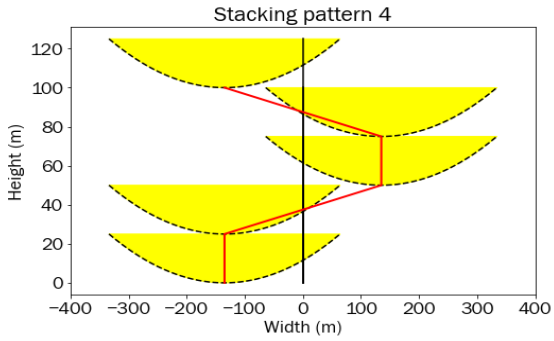
```
def parabola_fill(half_width,height,y_offset=0,x_offset=0):
    xaxis=range(-(half_width-1),half_width)
    xaxis=np.array(xaxis)+x_offset
    yaxis=(height/(half_width-1)**2)*((xaxis-x_offset)**2)+y_offset
    x_top_parabola=[min(xaxis), max(xaxis)]
    y_top_parabola=[max(yaxis), max(yaxis)]
```

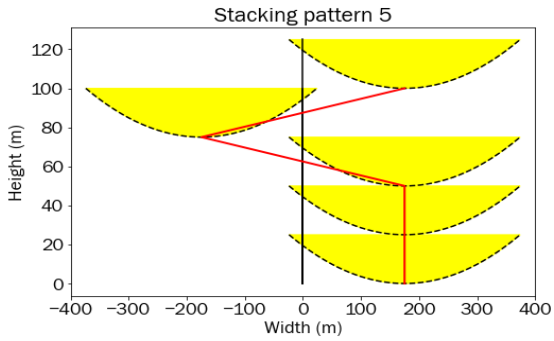
```
    ms_trace=max(yaxis)
    return plt.plot(xaxis,yaxis,'--',color='black'),plt.xlabel('Width (m)'), plt.ylabel('Thick
ness (m)'),plt.fill(xaxis,yaxis,'--',color='yellow'),plt.plot([0,0], [0,ms_trace],color='black
')

y_offset_test=[0,25,50,75,100]
plt.figure(figsize=(15, 25))
for k in range(len(most_likely)):
    hor_disp=most_likely[k]
    plt.subplot(5, 2, k+1)
    for i in range(len(hor_disp)):
        parabola_fill(half_width=200,height=25,y_offset=y_offset_test[i],x_offset=hor_disp[i])
    plt.title('Stacking pattern '+str(k+1))
    plt.ylabel('Height (m)') #y label
    plt.xlabel('Width (m)') #x label
    plt.xlim([-400,400])
    plt.plot(most_likely[k].flatten(), y_offset_test,'red',linewidth=2)
plt.tight_layout()
```

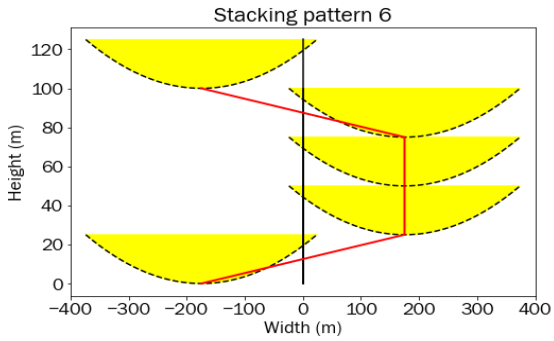

Stacking pattern 1
Height (m)
Width (m)

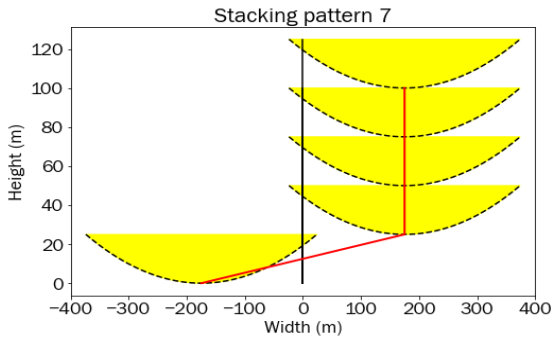
Stacking pattern 2
Height (m)
Width (m)

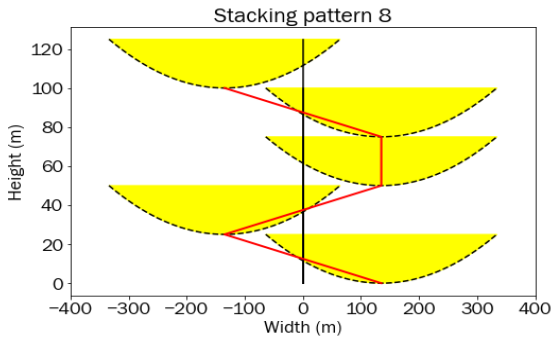

Stacking pattern 3
Height (m)
Width (m)

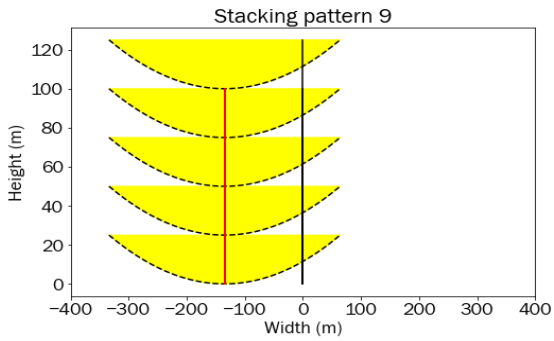

Stacking pattern 4
Height (m)
Width (m)

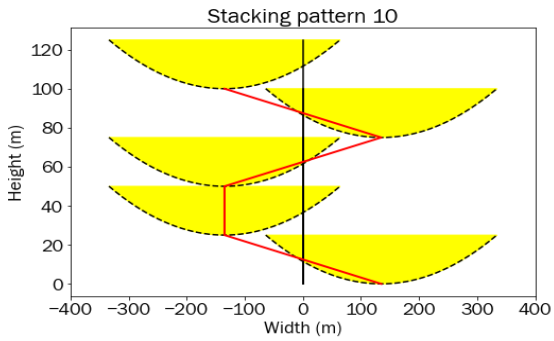

Stacking pattern 5
Height (m)
Width (m)


Stacking pattern 6
Height (m)
Width (m)


Stacking pattern 7
Height (m)
Width (m)


Stacking pattern 8
Height (m)
Width (m)


Stacking pattern 9
Height (m)
Width (m)


Stacking pattern 10
Height (m)
Width (m)

```
def distances(most_likely, y_offset_stack_pattern):
    total_magnitudes=[]
    for i in range(len(most_likely)):
        lat_offset=most_likely[i]
        ver_offset=y_offset_stack_pattern
        magnitude=[]
        for k in range(len(lat_offset)-1):
            distance=round((((lat_offset[k+1]-lat_offset[k])**2)+((ver_offset[k+1]-ver_offset[
k])**2))**(1/2),2)
            magnitude.append(distance)
        total_magnitudes.append(magnitude)
    return total_magnitudes

total_distances=distances(most_likely, y_offset_test)
pd.DataFrame(np.array(total_distances).sum(axis=1).reshape(10,1))
```

```
        0
0  100.00
1  592.30
2  751.78
3  592.30
4  751.78
5  751.78
6  425.89
7  838.45
8  100.00
9  838.45
```

Less likely stacking

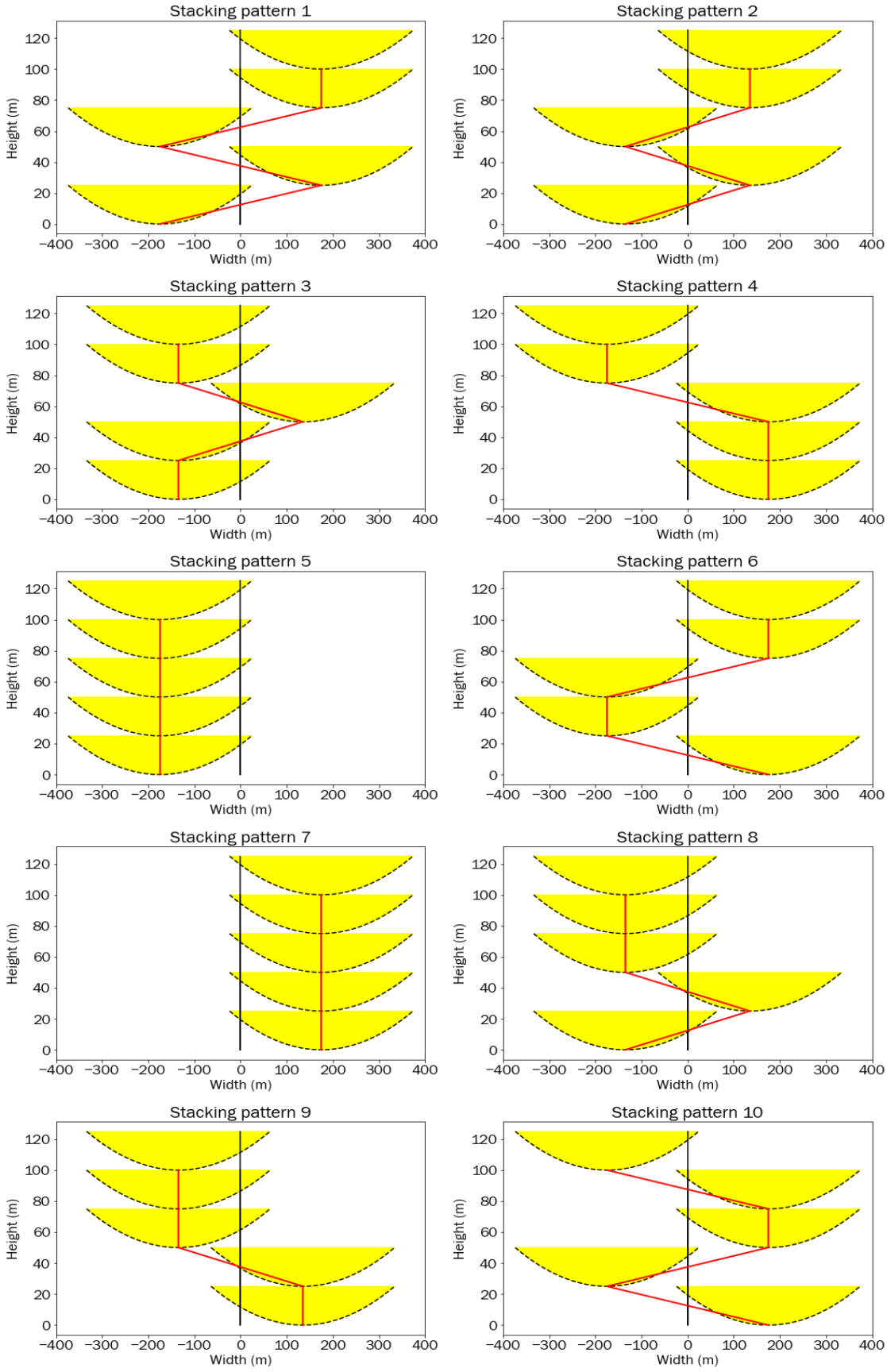
```
# Count Unique STacking Patterns
countrows = trans_prob_stack_pat.pivot_table(index = ['Elem1', 'Elem2', 'Elem3', 'Elem4', 'Ele
m5'], aggfunc = 'size')
countrows = pd.DataFrame(countrows)
countrows.columns = ['count']
countrows = countrows.reindex(countrows.sort_values(by='count', ascending=False).index)
countrows.tail(10)
```

```
                                  count
Elem1  Elem2  Elem3  Elem4  Elem5
-175.0  175.0 -175.0  175.0  175.0   1004
-135.0  135.0 -135.0  135.0  135.0   1000
       -135.0  135.0 -135.0 -135.0   1000
 175.0  175.0  175.0 -175.0 -175.0    999
-175.0 -175.0 -175.0 -175.0 -175.0    996
 175.0 -175.0 -175.0  175.0  175.0    993
        175.0  175.0  175.0  175.0    992
-135.0  135.0 -135.0 -135.0 -135.0    991
 135.0  135.0 -135.0 -135.0 -135.0    973
 175.0 -175.0  175.0  175.0 -175.0    966
```

```
less_likely=likelihood[-10:,:-1]
less_likely, less_likely.shape
```

```
(array([[-175.,  175., -175.,  175.,  175.],
        [-135.,  135., -135.,  135.,  135.],
        [-135., -135.,  135., -135., -135.],
        [ 175.,  175.,  175., -175., -175.],
        [-175., -175., -175., -175., -175.],
        [ 175., -175., -175.,  175.,  175.],
        [ 175.,  175.,  175.,  175.,  175.],
```

```
        [-135.,  135., -135., -135., -135.],
        [ 135.,  135., -135., -135., -135.],
        [ 175., -175.,  175.,  175., -175.]]),
 (10, 5))

y_offset_test=[0,25,50,75,100]
plt.figure(figsize=(15, 25))
for k in range(len(less_likely)):
    hor_disp=less_likely[k]
    plt.subplot(5, 2, k+1)
    for i in range(len(hor_disp)):
        parabola_fill(half_width=200,height=25,y_offset=y_offset_test[i],x_offset=hor_disp[i])
    plt.title('Stacking pattern '+str(k+1))
    plt.ylabel('Height (m)') #y label
    plt.xlabel('Width (m)') #x label
    plt.xlim([-400,400])
    plt.plot(less_likely[k].flatten(), y_offset_test,'red',linewidth=2)
plt.tight_layout()
```


Stacking pattern 1
Stacking pattern 2
Stacking pattern 3
Stacking pattern 4
Stacking pattern 5
Stacking pattern 6
Stacking pattern 7
Stacking pattern 8
Stacking pattern 9
Stacking pattern 10
Height (m)
Width (m)
0
20
40
60
80
100
120
−400
−300
−200
−100
0
100
200
300
400

```
total_distances=distances(less_likely, y_offset_test)
pd.DataFrame(np.array(total_distances).sum(axis=1).reshape(10,1))
```

```
         0
0  1077.67
1   838.45
2   592.30
3   425.89
4   100.00
5   751.78
6   100.00
7   592.30
8   346.15
9  1077.67
```

- Chapter 6

Uploading the data

```
import pandas as pd
import matplotlib.pyplot as plt
import NumPy as np

from matplotlib import rcParams
rcParams['font.family'] = 'sans-serif'
rcParams['font.sans-serif'] = ['Franklin Gothic Book']
rcParams['font.size'] = '18'

df=pd.read_excel('NN_Results_2.xlsx')
df
```

```
     Axis_Prob  Off_Axis_Prob  Margin_Prob  Pred_Class  True_Class  Geobody  \
0     0.200129       0.764206     0.035665           2           2        2
1     0.896818       0.103155     0.000027           1           1        3
2     0.185925       0.479642     0.334434           2           2        6
3     0.001238       0.096551     0.902211           3           3        7
4     0.217810       0.781036     0.001154           2           2        8
..         ...            ...          ...         ...         ...      ...
149   0.176377       0.722527     0.101096           2           2        3
150   0.985616       0.014327     0.000057           1           1        8
151   0.565369       0.434623     0.000009           1           1       11
152   0.000003       0.009516     0.990481           3           3       11
153   0.030788       0.920465     0.048747           2           3       11

    Meas_Sect Complex Set  Thickness
0       CACH1     'Lower'   8.126835
1       CACH1     'Lower'  17.730909
2       CACH1     'Lower'   6.141687
3       CACH1     'Lower'  11.450495
4       CACH1     'Lower'  18.390356
..        ...         ...        ...
149       VV2     'Upper'  11.824580
150       VV2     'Upper'  11.138141
151       VV7     'Upper'  26.417171
152    VVEDGE     'Upper'   5.896554
153      VVWB     'Upper'  14.202635

[154 rows x 9 columns]
```

```
Meas_Sect=df[df['Meas_Sect']=='CACH1'] #REPLACE THE NAME OF THE MEASURED SECTION
Meas_Sect=Meas_Sect[['Axis_Prob', 'Off_Axis_Prob', 'Margin_Prob']]
Meas_Sect
```

```
   Axis_Prob  Off_Axis_Prob  Margin_Prob
0   0.200129       0.764206     0.035665
1   0.896818       0.103155     0.000027
2   0.185925       0.479642     0.334434
3   0.001238       0.096551     0.902211
4   0.217810       0.781036     0.001154
```

```
Meas_Sect=Meas_Sect.values
Meas_Sect
```

```
array([[2.00128968e-01, 7.64206174e-01, 3.56648580e-02],
       [8.96817877e-01, 1.03154714e-01, 2.74090607e-05],
       [1.85924709e-01, 4.79641650e-01, 3.34433641e-01],
       [1.23753958e-03, 9.65514118e-02, 9.02211049e-01],
       [2.17809653e-01, 7.81036309e-01, 1.15403762e-03]])
```

Histograms (or Probability Density Functions, PDFs) and Cummulative Distribution Functions, or CDFs

```
def prob(Meas_Sect):
    prob=np.insert(Meas_Sect, 0, 0, 1)
    return prob

def prob_cumsum(Meas_Sect):
    prob_cumsum=Meas_Sect.cumsum(axis=1)
    prob_cumsum=np.insert(prob_cumsum, 0, 0, 1)
    return prob_cumsum

Meas_Sect_probs=prob(Meas_Sect)
Meas_Sect_probs, Meas_Sect_probs.shape
```

```
(array([[0.00000000e+00, 2.00128968e-01, 7.64206174e-01, 3.56648580e-02],
        [0.00000000e+00, 8.96817877e-01, 1.03154714e-01, 2.74090607e-05],
        [0.00000000e+00, 1.85924709e-01, 4.79641650e-01, 3.34433641e-01],
        [0.00000000e+00, 1.23753958e-03, 9.65514118e-02, 9.02211049e-01],
        [0.00000000e+00, 2.17809653e-01, 7.81036309e-01, 1.15403762e-03]]),
 (5, 4))
```

```
Meas_Sect_probs_cum=prob_cumsum(Meas_Sect)
Meas_Sect_probs_cum, Meas_Sect_probs_cum.shape
```

```
(array([[0.        , 0.20012897, 0.96433514, 1.        ],
        [0.        , 0.89681788, 0.99997259, 1.        ],
        [0.        , 0.18592471, 0.66556636, 1.        ],
        [0.        , 0.00123754, 0.09778895, 1.        ],
        [0.        , 0.21780965, 0.99884596, 1.        ]]),
 (5, 4))
```

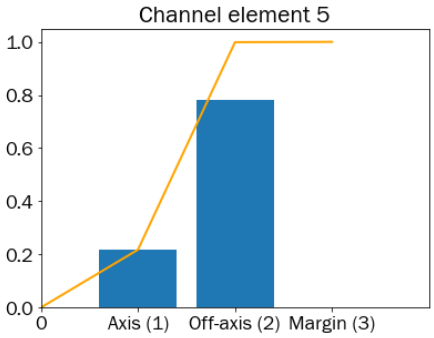
```
ArchitecturalElementPositionNames=['0', 'Axis (1)', 'Off-axis (2)', 'Margin (3)']
ArchitecturalElementPositionNames
```

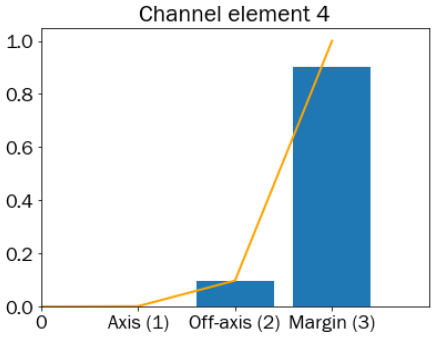
```
['0', 'Axis (1)', 'Off-axis (2)', 'Margin (3)']
```

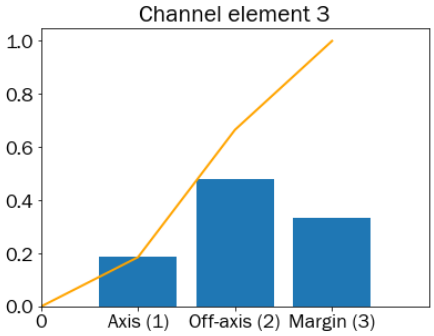
```
ArchitecturalElementPositionNumbers=np.array([0, 1, 2, 3])
print(ArchitecturalElementPositionNumbers.shape, ArchitecturalElementPositionNumbers)
```

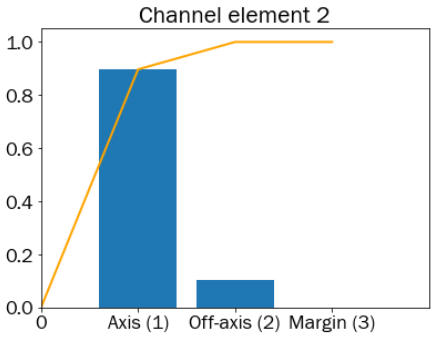
```
(4,) [0 1 2 3]
```

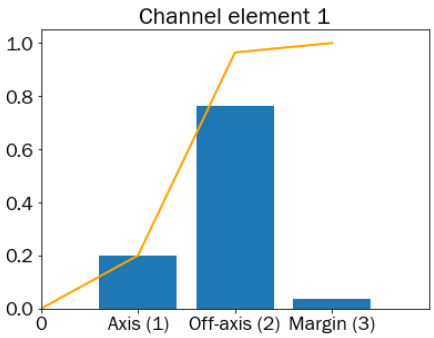
```
def pdfs_and_cdfs(Meas_Sect_probs, Meas_Sect_probs_cum):
    plt.figure(figsize=(6, 22)) #create the figure, then change the size of it
    for c in range(len(Meas_Sect_probs)): #for each sample in the dimension length... THIS IS
USEFUL! TO GO OVER INDEXES OR REPEAT X TIMES A SEQUENCE!!!
        plt.subplot(len(Meas_Sect_probs), 1, len(Meas_Sect_probs)-c) #display the graphs in 4
rows, 3 columns
        plt.bar(ArchitecturalElementPositionNames, Meas_Sect_probs[c]) #plot bars
        plt.plot(Meas_Sect_probs_cum[c], color='orange', linewidth=2) #plot a line above
        plt.title('Channel element '+str(c+1))
        plt.ylim((0,1.05)) #same dimensions for all the graphs
        plt.xlim((0,4)) #same dimensions for all the graphs
    plt.tight_layout() #improve the layout

pdfs_and_cdfs(Meas_Sect_probs, Meas_Sect_probs_cum)
```


Channel element 5
1.0
0.8
0.6
0.4
0.2
0.0
0
Axis (1)
Off-axis (2)
Margin (3)


Channel element 4
1.0
0.8
0.6
0.4
0.2
0.0
0
Axis (1)
Off-axis (2)
Margin (3)

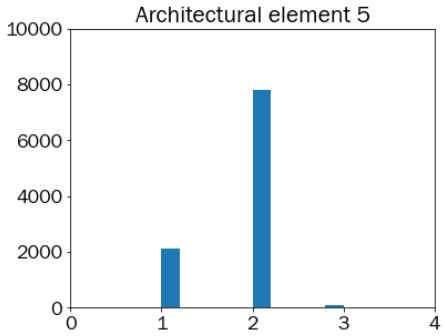

Channel element 3
1.0
0.8
0.6
0.4
0.2
0.0
0
Axis (1)
Off-axis (2)
Margin (3)

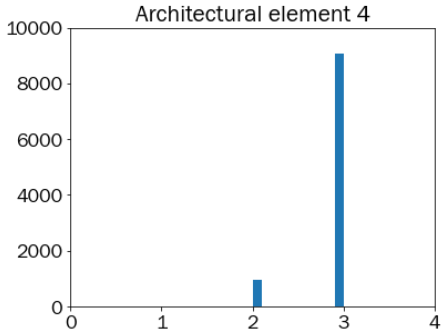

Channel element 2
1.0
0.8
0.6
0.4
0.2
0.0
0
Axis (1)
Off-axis (2)
Margin (3)

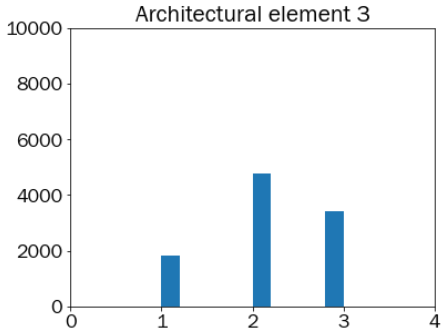

Channel element 1
1.0
0.8
0.6
0.4
0.2
0.0
0
Axis (1)
Off-axis (2)
Margin (3)

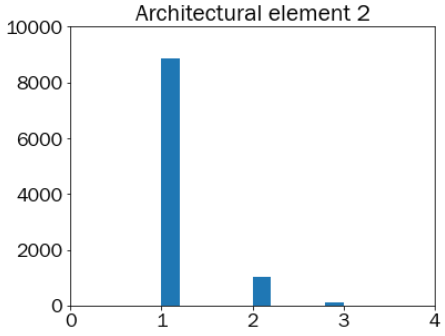
```
import random

def montecarlo(NameMS, NameMS_cumsum, n_samples):
    arch_pos=[0,1,2,3]
    results=[]
    plt.figure(figsize=(6, 22))
    for probs in range(len(NameMS)):
        #EMPTY LIST 2: classifications from eCDF (1,2,3)
        mc_sim=[]
        a,b,c,d=arch_pos
        w,x,y,z=NameMS_cumsum[probs]
        plt.subplot(len(NameMS), 1, len(NameMS)-probs)
        #repeat n samples (n_samples)
        for n in range(n_samples):
            value=round(np.random.uniform(0, 1), ndigits=2)
            if w < value <= x:
                arch_el = b
            elif x < value <= y:
                arch_el = c
            else:
                arch_el = d
            # ___OUTPUT 2___
            mc_sim.append(arch_el)
        p_axis=round(mc_sim.count(b)/n_samples, ndigits=2)
        p_offaxis=round(mc_sim.count(c)/n_samples, ndigits=2)
        p_margin=round(mc_sim.count(d)/n_samples, ndigits=2)
        #___OUTPUT 1____
        results.append([p_axis, p_offaxis, p_margin])
        plt.hist(mc_sim)
        plt.title('Architectural element ' + str(probs+1))
        #plt.ylabel('Number of samples',fontsize=16) #y label
        #plt.xlabel('Architectural position',fontsize=16) #x label
        plt.ylim((0,n_samples)) #same dimensions for all the graphs
        plt.xlim((0,4)) #same dimensions for all the graphs
    plt.tight_layout()
    return results, type(results), pd.DataFrame(results,columns=['Axis', 'Off-axis', 'Margin']
)

montecarlo_results=montecarlo(Meas_Sect_probs, Meas_Sect_probs_cum, 10000)
montecarlo_results
```

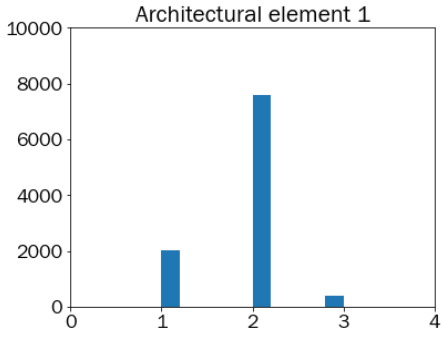
```
([[0.2, 0.76, 0.04],
  [0.89, 0.1, 0.01],
  [0.18, 0.48, 0.34],
  [0.0, 0.09, 0.91],
  [0.21, 0.78, 0.01]],
 list,
    Axis  Off-axis  Margin
 0  0.20      0.76    0.04
 1  0.89      0.10    0.01
 2  0.18      0.48    0.34
 3  0.00      0.09    0.91
 4  0.21      0.78    0.01)
```


Architectural element 5
10000
8000
6000
4000
2000
0
0
1
2
3
4


Architectural element 4
10000
8000
6000
4000
2000
0
0
1
2
3
4


Architectural element 3
10000
8000
6000
4000
2000
0
0
1
2
3
4


Architectural element 2
10000
8000
6000
4000
2000
0
0
1
2
3
4

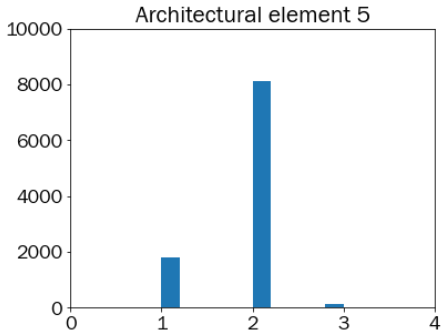

Architectural element 1
10000
8000
6000
4000
2000
0
0
1
2
3
4

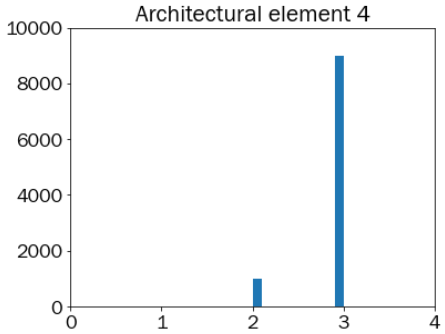
Transitional probabilities incorporation

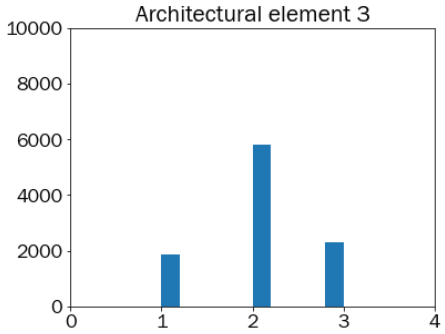
```
#AFTER CALCULATING THE TRANSITIONAL PROBABILITIES FROM SCRIPT 2
trans_prob=np.array([0.29411764705882354,
 0.5588235294117647,
 0.14705882352941177,
 0.380952380952381,
 0.42857142857142855,
 0.1904761904761905,
 0.36,
 0.32,
 0.32]).reshape(3,3).round(2)
trans_prob

array([[0.29, 0.56, 0.15],
       [0.38, 0.43, 0.19],
       [0.36, 0.32, 0.32]])

def montecarlo_trans_prob(NameMS, NameMS_cumsum, n_samples): #DOUBLE MONTECARLO SIMULATION
    simulation=[]
    plt.figure(figsize=(6, 22))
    for probs in range(len(NameMS)):
        mc_sim=[]
        #a,b,c,d=arch_pos
        w,x,y,z=NameMS_cumsum[probs]
        initial_probs=NameMS[probs,1:]
        plt.subplot(len(NameMS), 1, len(NameMS)-probs)
        for i in range(n_samples):
            value1=round(np.random.uniform(0, 1), ndigits=2)
            if w < value1 <= x:
                trans_prob_imp=initial_probs*trans_prob[0]
                norm_probs=trans_prob_imp/sum(trans_prob_imp)
                norm_cumsum=np.insert(norm_probs.cumsum(), 0, 0, 0)
                l,m,n,o=norm_cumsum
                value2=round(np.random.uniform(0, 1), ndigits=2)
                if l < value2 <= m:
                    arch_ele = 1
                elif m < value2 <= n:
                    arch_ele = 2
                else:
                    arch_ele = 3
            elif x < value1 <= y:
                trans_prob_imp=initial_probs*trans_prob[1]
                norm_probs=trans_prob_imp/sum(trans_prob_imp)
                norm_cumsum=np.insert(norm_probs.cumsum(), 0, 0, 0)
                l,m,n,o=norm_cumsum
                value2=round(np.random.uniform(0, 1), ndigits=2)
                if l < value2 <= m:
                    arch_ele = 1
                elif m < value2 <= n:
                    arch_ele = 2
                else:
                    arch_ele = 3
            else:
                trans_prob_imp=initial_probs*trans_prob[2]
                norm_probs=trans_prob_imp/sum(trans_prob_imp)
                norm_cumsum=np.insert(norm_probs.cumsum(), 0, 0, 0)
                l,m,n,o=norm_cumsum
                value2=round(np.random.uniform(0, 1), ndigits=2)
                if l < value2 <= m:
                    arch_ele = 1
```

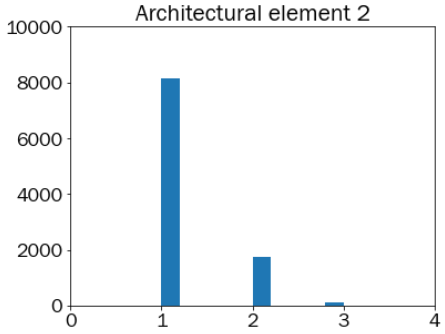
```
                elif m < value2 <= n:
                    arch_ele = 2
                else:
                    arch_ele = 3
            mc_sim.append(arch_ele)
            p_axis=round(mc_sim.count(1)/n_samples, ndigits=2)
            p_offaxis=round(mc_sim.count(2)/n_samples, ndigits=2)
            p_margin=round(mc_sim.count(3)/n_samples, ndigits=2)
        simulation.append([p_axis, p_offaxis, p_margin])
        plt.hist(mc_sim)
        plt.title('Architectural element ' + str(probs+1))
        #plt.ylabel('Number of samples',fontsize=16) #y label
        #plt.xlabel('Architectural position',fontsize=16) #x label
        plt.ylim((0,n_samples)) #same dimensions for all the graphs
        plt.xlim((0,4)) #same dimensions for all the graphs
    plt.tight_layout()
    return simulation, type(simulation), pd.DataFrame(simulation,columns=['Axis', 'Off-axis',
'Margin'])

double_montecarlo=montecarlo_trans_prob(Meas_Sect_probs, Meas_Sect_probs_cum, 10000)
double_montecarlo
```

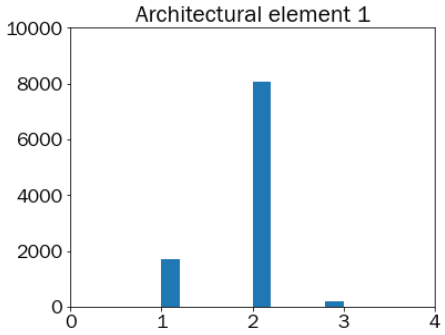
```
([[0.17, 0.81, 0.02],
  [0.81, 0.17, 0.01],
  [0.19, 0.58, 0.23],
  [0.0, 0.1, 0.9],
  [0.18, 0.81, 0.01]],
 list,
    Axis  Off-axis  Margin
 0  0.17      0.81    0.02
 1  0.81      0.17    0.01
 2  0.19      0.58    0.23
 3  0.00      0.10    0.90
 4  0.18      0.81    0.01)
```


Architectural element 5
10000
8000
6000
4000
2000
0
0
1
2
3
4


Architectural element 4
10000
8000
6000
4000
2000
0
0
1
2
3
4


Architectural element 3
10000
8000
6000
4000
2000
0
0
1
2
3
4


Architectural element 2
10000
8000
6000
4000
2000
0
0
1
2
3
4


Architectural element 1
10000
8000
6000
4000
2000
0
0
1
2
3
4

```
a,b,c=double_montecarlo
results=a
results
```

```
[[0.17, 0.81, 0.02],
 [0.81, 0.17, 0.01],
 [0.19, 0.58, 0.23],
 [0.0, 0.1, 0.9],
 [0.18, 0.81, 0.01]]
```

Getting the maximum probabilities

```
arch_pos_codes={1:'axis',
                2:'off-axis',
                3:'margin'}
arch_pos_codes
```

```
{1: 'axis', 2: 'off-axis', 3: 'margin'}
```

```
max_prob=np.amax(results, axis=1)
max_prob, max_prob.shape, type(max_prob)
```

```
(array([0.81, 0.81, 0.58, 0.9 , 0.81]), (5,), NumPy.ndarray)
```

```
max_final_facies=np.argmax(results,axis=1)+1
max_final_facies, max_final_facies.shape, type(max_final_facies)
```

```
(array([2, 1, 2, 3, 2], dtype=int64), (5,), NumPy.ndarray)
```

```
ms_final_results_codes_max=[]
for i in range(len(results)):
    codes=arch_pos_codes[max_final_facies[i]]
    ms_final_results_codes_max.append(codes)
ms_final_results_codes_max=np.array(ms_final_results_codes_max)
ms_final_results_codes_max
```

```
array(['off-axis', 'axis', 'off-axis', 'margin', 'off-axis'], dtype='<U8')
```

```
ms_final_results_max=np.stack((max_final_facies, ms_final_results_codes_max, max_prob), axis=1
)
ms_final_results_max, ms_final_results_max.shape, type(ms_final_results_max)
```

```
(array([['2', 'off-axis', '0.81'],
        ['1', 'axis', '0.81'],
        ['2', 'off-axis', '0.58'],
        ['3', 'margin', '0.9'],
        ['2', 'off-axis', '0.81']], dtype='<U32'),
 (5, 3),
 NumPy.ndarray)
```

Getting the minimum probabilities

```
min_prob=np.amin(results, axis=1)
min_prob, min_prob.shape, type(min_prob)
```

```
(array([0.02, 0.01, 0.19, 0.  , 0.01]), (5,), NumPy.ndarray)
```

```
min_final_facies=np.argmin(results,axis=1)+1
min_final_facies, min_final_facies.shape, type(min_final_facies)
```

```
(array([3, 3, 1, 1, 3], dtype=int64), (5,), NumPy.ndarray)
```

```
ms_final_results_codes_min=[]
for i in range(len(results)):
    codes=arch_pos_codes[min_final_facies[i]]
    ms_final_results_codes_min.append(codes)
ms_final_results_codes_min=np.array(ms_final_results_codes_min)
ms_final_results_codes_min
```

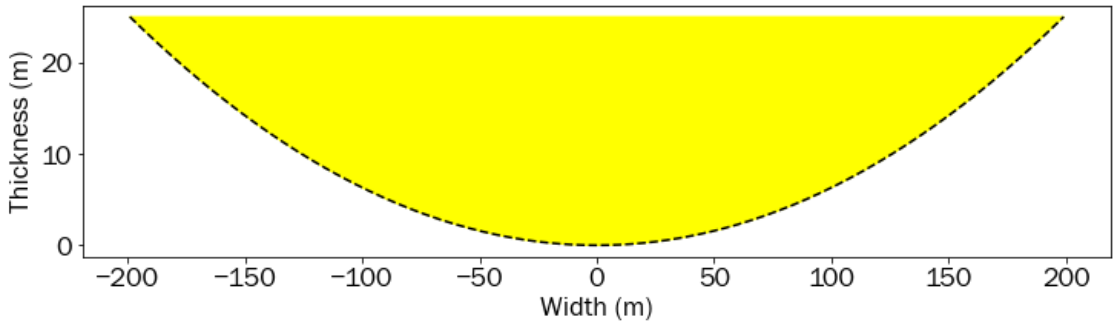
```
array(['margin', 'margin', 'axis', 'axis', 'margin'], dtype='<U6')
```

```
ms_final_results_min=np.stack((min_final_facies, ms_final_results_codes_min, min_prob), axis=1
)
ms_final_results_min, ms_final_results_min.shape, type(ms_final_results_min)
```

```
(array([['3', 'margin', '0.02'],
        ['3', 'margin', '0.01'],
        ['1', 'axis', '0.19'],
        ['1', 'axis', '0.0'],
        ['3', 'margin', '0.01']], dtype='<U32'),
 (5, 3),
 NumPy.ndarray)
```

Parabola generation

```
def parabola_fill(half_width,height,y_offset=0,x_offset=0):
    xaxis=range(-(half_width-1),half_width)
    xaxis=np.array(xaxis)+x_offset
    yaxis=(height/(half_width-1)**2)*((xaxis-x_offset)**2)+y_offset
    x_top_parabola=[min(xaxis), max(xaxis)]
    y_top_parabola=[max(yaxis), max(yaxis)]
    ms_trace=max(yaxis)
    return plt.figure(figsize=(12, 3)),plt.plot(xaxis,yaxis,'--',color='black'),plt.xlabel('Wi
dth (m)'), plt.ylabel('Thickness (m)'),plt.fill(xaxis,yaxis,'--',color='yellow')

par_fill=parabola_fill(half_width=200,height=25,y_offset=0,x_offset=0)
```



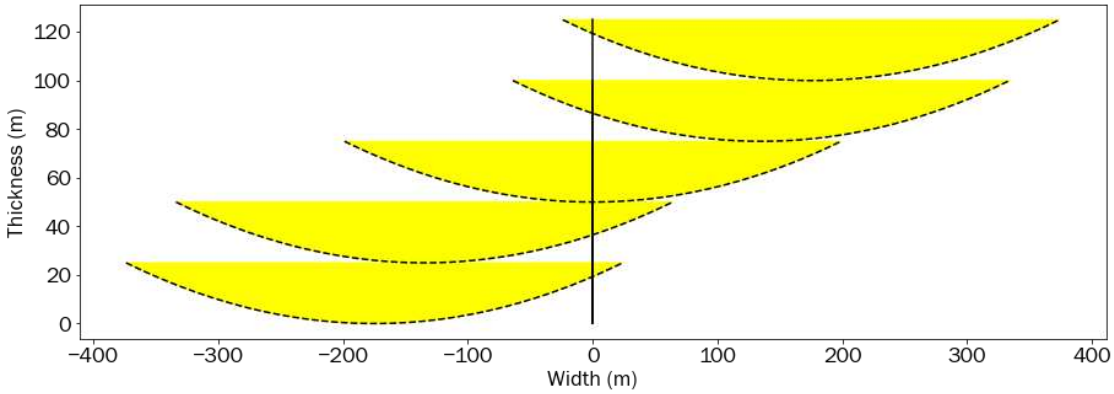
Setting the parameters for the five possible architectural positions

```
width_total=400
perc_to_axis=0
perc_to_offaxis=0.3375
perc_to_margin=0.4375
temp_x_offset=[width_total*-perc_to_margin, width_total*-perc_to_offaxis, width_total*perc_to_
axis, width_total*perc_to_offaxis, width_total*perc_to_margin]
y_offset=25
temp_y_offset=[]
for i in range(len(temp_x_offset)):
    step=y_offset*i
    temp_y_offset.append(step)
```

```
print(temp_x_offset, type(temp_x_offset), len(temp_x_offset))
print(temp_y_offset, type(temp_y_offset), len(temp_y_offset))
```

```
[-175.0, -135.0, 0, 135.0, 175.0] <class 'list'> 5
[0, 25, 50, 75, 100] <class 'list'> 5
```

```
def parabola_fill(half_width,height,y_offset=0,x_offset=0):
    xaxis=range(-(half_width-1),half_width)
    xaxis=np.array(xaxis)+x_offset
    yaxis=(height/(half_width-1)**2)*((xaxis-x_offset)**2)+y_offset
    x_top_parabola=[min(xaxis), max(xaxis)]
    y_top_parabola=[max(yaxis), max(yaxis)]
    ms_trace=max(yaxis)
    return plt.plot(xaxis,yaxis,'--',color='black'),plt.xlabel('Width (m)'), plt.ylabel('Thick
ness (m)'),plt.fill(xaxis,yaxis,'--',color='yellow'),plt.plot([0,0], [0,ms_trace],color='black
')

plt.figure(figsize=(15, 5))
for i in range(len(temp_x_offset)):
    parabola_fill(half_width=200,height=25,y_offset=temp_y_offset[i],x_offset=temp_x_offset[i]
)
```



Generation of stacking patterns

```
ms_final_results_max, ms_final_results_max.shape, type(ms_final_results_max)
```

```
(array([['2', 'off-axis', '0.81'],
        ['1', 'axis', '0.81'],
        ['2', 'off-axis', '0.58'],
        ['3', 'margin', '0.9'],
        ['2', 'off-axis', '0.81']], dtype='<U32'),
 (5, 3),
 NumPy.ndarray)
```

Generating stacking patterns that match to thickness

```
ms_final_results_max
```

```
array([['2', 'off-axis', '0.81'],
       ['1', 'axis', '0.81'],
       ['2', 'off-axis', '0.58'],
       ['3', 'margin', '0.9'],
       ['2', 'off-axis', '0.81']], dtype='<U32')
```

```
thickness=df[df['Meas_Sect']=='CACH1']['Thickness'] #NAME OF THE MEASURED SECTION
thickness=thickness.values
thickness
```

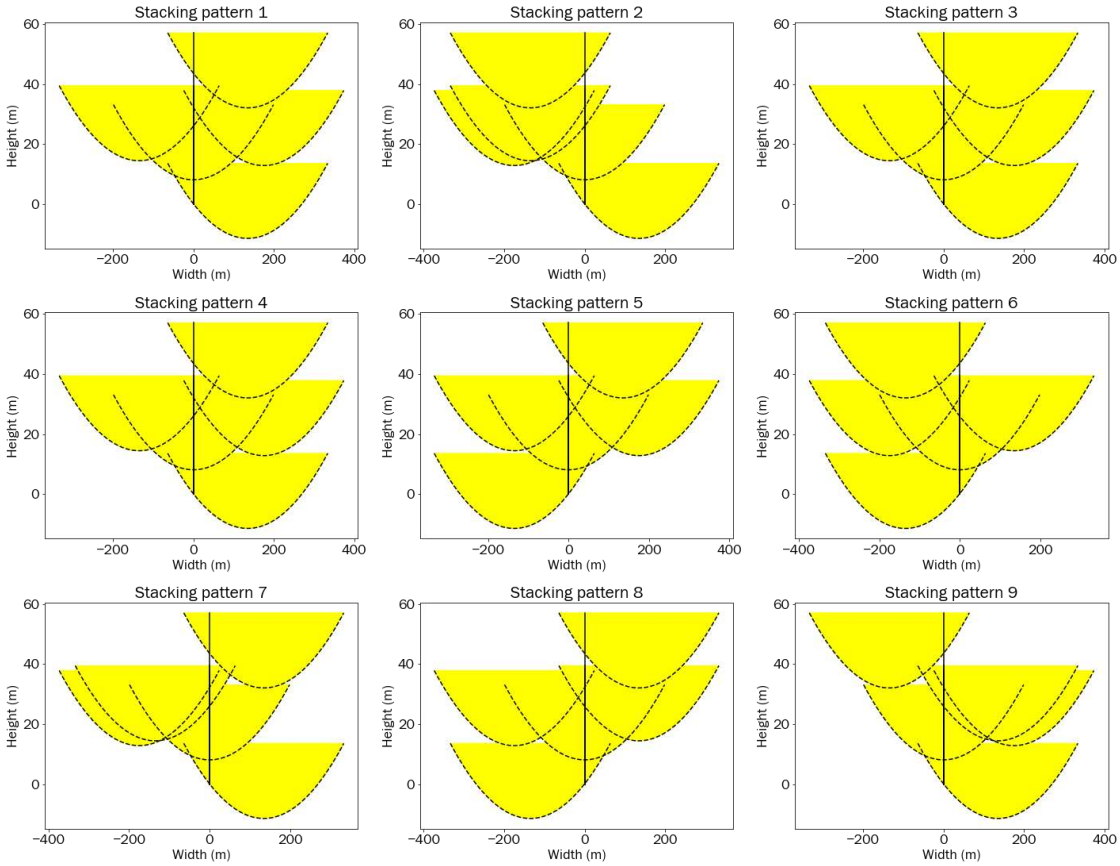
```
array([ 8.12683487, 17.73090935,  6.14168692, 11.45049477, 18.39035606])
```

```
def y_offset_to_thickness(results, thicknesses, height=25, width_total=400, perc_to_axis=0, pe
rc_to_offaxis=0.3375, perc_to_margin=0.4375):
    net_erosion=[]
    x_offset=[width_total*perc_to_axis, width_total*perc_to_offaxis, width_total*perc_to_margi
n]
    for c in x_offset:
        value=height/(width_total/2)**2*c**2
        net_erosion.append(value)
    net_y_offset=[]
    arch_pos=results[:,0]
    for i in range(len(arch_pos)):
        if int(arch_pos[i])==1:
            value=net_erosion[0]
        else:

            if int(arch_pos[i])==2:
                value=net_erosion[1]
            else:
                value=net_erosion[2]
        net_y_offset.append(value)
    net_y_offset=np.array(net_y_offset)
    thickness_cumsum=thicknesses.cumsum()
    thickness_cumsum=np.insert(thickness_cumsum[:-1], 0, 0)
    final_y_offset=(thickness_cumsum-net_y_offset).tolist()
    return final_y_offset

y_offset=y_offset_to_thickness(ms_final_results_max, thickness)
y_offset
```

```
[-11.390625,
 8.12683486938477,
 14.46711921691897,
 12.858806133270292,
 32.05930089950569]
```

```
def stacking_patterns_to_thick(results, n_stack_patterns, net_y_offset, width_total=400, perc_
to_axis=0, perc_to_offaxis=0.3375, perc_to_margin=0.4375):
    temp_incision=[width_total*-perc_to_margin, width_total*-perc_to_offaxis, width_total*perc
_to_axis, width_total*perc_to_offaxis, width_total*perc_to_margin]
    x_offset_list=[]
    for j in range(n_stack_patterns):
        incision=[]
        for i in range(len(results)):
            rand_margin=np.random.choice([-2,2])
            rand_offaxis=np.random.choice([-1,1])
            rand_axis=0
            if rand_margin==-2 and int(results[i,0])==3:
                inc_step=temp_incision[0]
            else:
                if rand_offaxis==-1 and int(results[i,0])==2:
                    inc_step=temp_incision[1]
                else:
                    if rand_axis==0 and int(results[i,0])==1:
                        inc_step=temp_incision[2]
                    else:
```

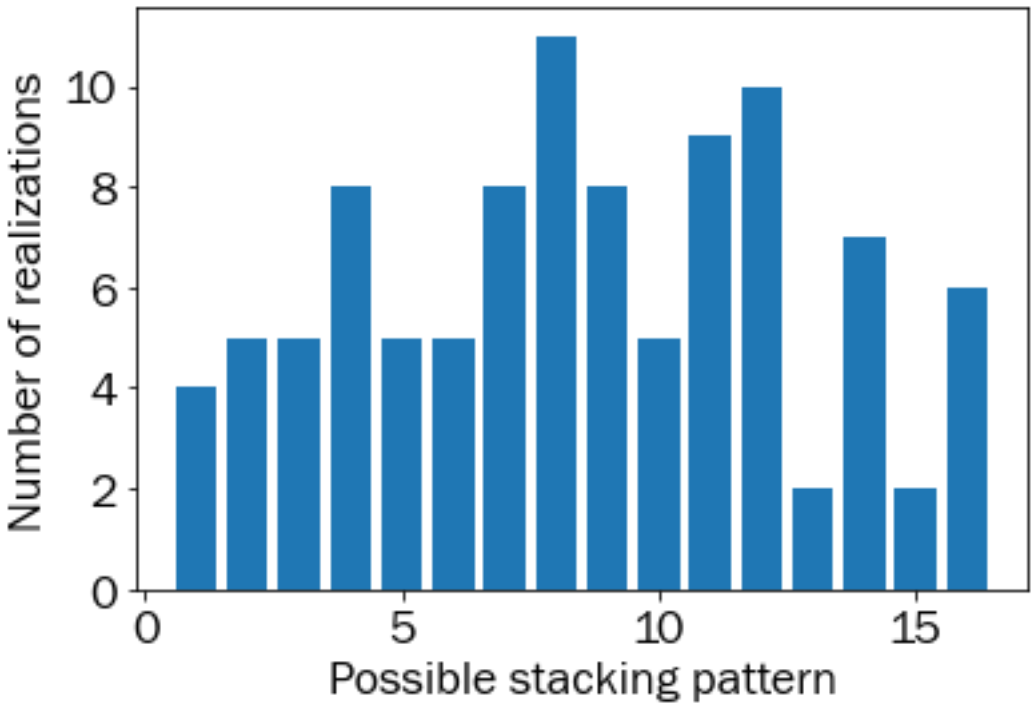
```
                        if rand_offaxis==1 and int(results[i,0])==2:
                            inc_step=temp_incision[3]
                        else:
                            inc_step=temp_incision[4]
                incision.append(inc_step)
            x_offset_list.append(incision)
        x_offset_list=np.array(x_offset_list)
        y_offset_list=net_y_offset
        #y_offset_list=[]
        #for k in range(len(incision)):
            #agg_step=y_offset*k
            #y_offset_list.append(agg_step)
        return x_offset_list, y_offset_list

stack_pat_test=stacking_patterns_to_thick(ms_final_results_max, 9, y_offset)
x_offset, y_offset=stack_pat_test

y_offset

[-11.390625,
 8.12683486938477,
 14.46711921691897,
 12.858806133270292,
 32.05930089950569]

plt.figure(figsize=(20, 20))
for k in range(len(x_offset)):
    hor_disp=x_offset[k]
    plt.subplot(4, 3, k+1)
    for i in range(len(hor_disp)):
        parabola_fill(half_width=200,height=25,y_offset=y_offset[i],x_offset=hor_disp[i])
    plt.title('Stacking pattern '+str(k+1))
    plt.ylabel('Height (m)') #y label
    plt.xlabel('Width (m)') #x label
plt.tight_layout()
```



Generating possible stacking patterns with thicknesses

```
def multiple_stack_pat(x_offset):
    possibilities_list=np.unique(x_offset, axis=0).tolist()
    x_offset_list=x_offset.tolist()
    labels=list(range(1,len(possibilities_list)+1))
    prel_results=[]
    for i in range(len(possibilities_list)):
        counting=x_offset_list.count(possibilities_list[i])
        prel_results.append(counting)
    return plt.bar(labels, prel_results), plt.ylabel('Number of realizations'), plt.xlabel('Po
ssible stacking pattern')

stack_pat_15m_100=stacking_patterns_to_thick(ms_final_results_max, 100, y_offset)
n_incision, aggradation=stack_pat_15m_100
multiple_stack_pat(n_incision)
```

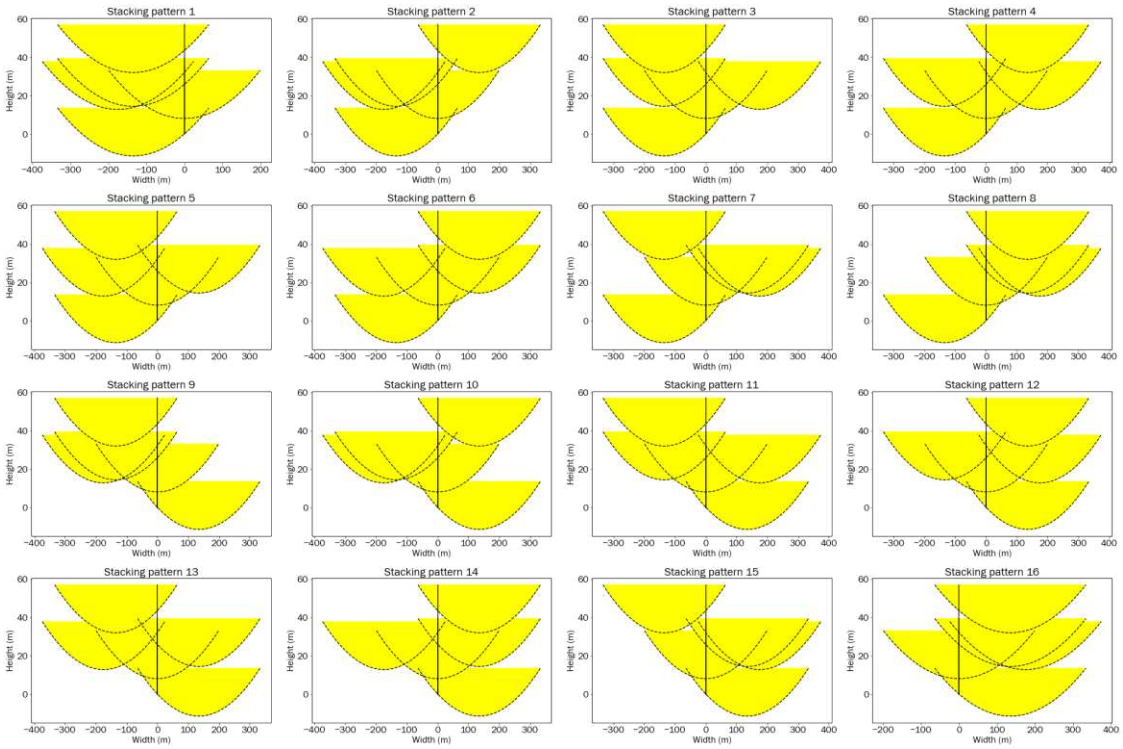
```
(<BarContainer object of 16 artists>,
 Text(0, 0.5, 'Number of realizations'),
 Text(0.5, 0, 'Possible stacking pattern'))
```




```
unique_possibilities=np.unique(n_incision, axis=0)
unique_possibilities, unique_possibilities.shape
```

```
(array([[-135.,    0., -135., -175., -135.],
        [-135.,    0., -135., -175.,  135.],
        [-135.,    0., -135.,  175., -135.],
        [-135.,    0., -135.,  175.,  135.],
        [-135.,    0.,  135., -175., -135.],
        [-135.,    0.,  135., -175.,  135.],
        [-135.,    0.,  135.,  175., -135.],
        [-135.,    0.,  135.,  175.,  135.],
        [ 135.,    0., -135., -175., -135.],
        [ 135.,    0., -135., -175.,  135.],
        [ 135.,    0., -135.,  175., -135.],
        [ 135.,    0., -135.,  175.,  135.],
        [ 135.,    0.,  135., -175., -135.],
        [ 135.,    0.,  135., -175.,  135.],
        [ 135.,    0.,  135.,  175., -135.],
        [ 135.,    0.,  135.,  175.,  135.]]),
 (16, 5))
```

```
plt.figure(figsize=(30, 20))
for k in range(len(unique_possibilities)):
    hor_disp=unique_possibilities[k]
    plt.subplot(4, 4, k+1)
    for i in range(len(hor_disp)):
        parabola_fill(half_width=200,height=25,y_offset=aggradation[i],x_offset=hor_disp[i])
    plt.title('Stacking pattern '+str(k+1))
    plt.ylabel('Height (m)') #y label
    plt.xlabel('Width (m)') #x label
plt.tight_layout()
```


Stacking pattern 1
Stacking pattern 2
Stacking pattern 3
Stacking pattern 4
Stacking pattern 5
Stacking pattern 6
Stacking pattern 7
Stacking pattern 8
Stacking pattern 9
Stacking pattern 10
Stacking pattern 11
Stacking pattern 12
Stacking pattern 13
Stacking pattern 14
Stacking pattern 15
Stacking pattern 16
Height (m)
Width (m)