UCLouvain

École polytechnique de Louvain

NomoPad: a dynamic nomography application for interactive computation on iPad

Author: **Guillaume Nizet**
Supervisor: **Peter Van Roy**
Readers: **Peter Van Roy, Jean Vanderdonckt, Magali Legast**
Academic year 2022–2023
Master [120] in Computer Science

Contents

Chapter 1

Introduction

1.1 History of Nomography

Nomography (from Greek *nomos*, "law" and *graphè*, "drawing") refers to the graphical representation of mathematical laws or relationships. This term was invented in 1891 by Maurice d'Ocagne. In his *Traité de Nomographie* [1], he introduced the concept of nomography as a way of representing an equation composed of several variables. He wanted to compare and standardize previous work of graphical equation representation, such as the works of Descartes [2], who laid the foundations of analytical geometry and introduced the planar representation of two-variable equations, and Pouchet [3], who first proposed a way to represent three-variable equations graphically. D'Ocagne's objective was to allow for straightforward reading of the different variable values composing an equation.

Once constructed, these graphical representations called **nomograms** show the relationships between the variables of an equation. Each variable is represented by a graduated line called **scale**. An **index line** is traced intersecting all the scales and the values of the variables can simply be read at the intersection points between the index line and each scale. As an example, Figure 1.2 shows the nomogram representing the following equation:

$$BD = \frac{AB.BC}{\sqrt{AB^2 + BC^2}}$$

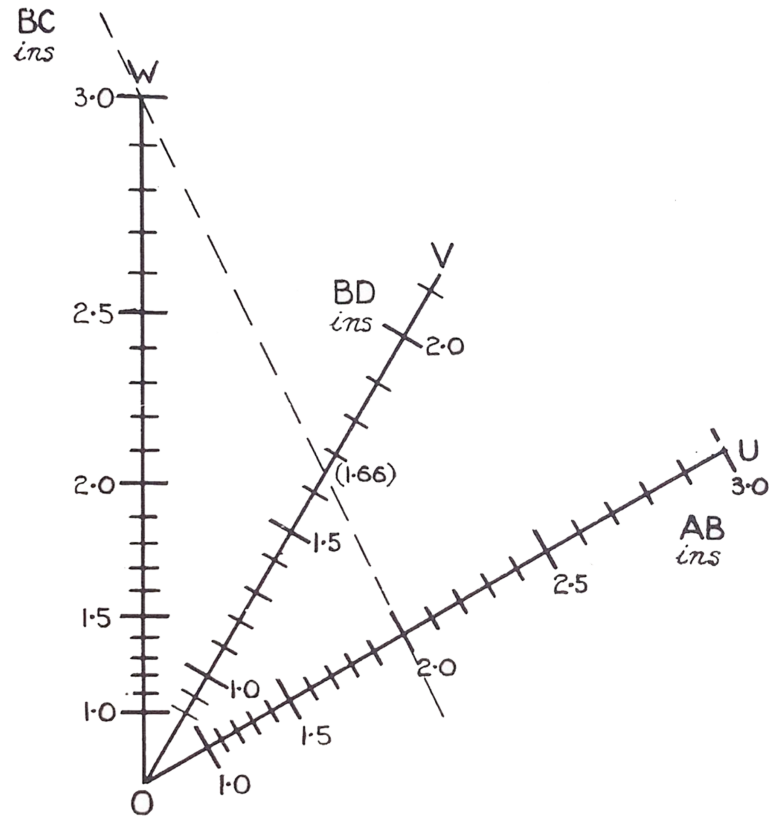

Figure 1.1: Example of a 3-variable nomogram [4]

The variables AB, BD and BC are represented respectively by the scales U, V and W. By laying the index line on the nomogram, the variable values can be estimated immediately at the intersection with the scales: in this example, with AB = 2 inches and BC = 3 inches, the value of BD can be estimated to around 1.66 inches.

This is a good illustration of the power of nomography: nomograms give a graphical intuition of the relationship between the variables and the resolution of the underlying equation becomes trivial. However, nomograms also have important limitations since they have to be drawn or printed on paper: the range of the scales cannot be infinite and the precision with which the variable values can be read depends on the size of the scales, which are limited by the size of the paper. Once the first scientific calculators were invented around 1970, offering the same speed of equation resolution as nomograms but with far better precision, nomography quickly became obsolete, its only advantage being the graphical intuition of the represented relationship.

1.2 Contributions

Nomography's limited practicality and lack of precision have largely contributed to its obsolescence. Nomograms, designed to visualize specific relationships, necessitate manual paper printing and lack the accuracy demanded by modern calculations. While visually intuitive, they fall short when compared to the computational capabilities of current scientific calculators. But in the last years, a new technology that could bring back nomography up to date is emerging: tablet computers, and more specifically iPads. Their computing power largely outperforms scientific calculators thanks to the latest Apple chips and they are equipped with high-definition screens that are perfectly suited for displaying nomograms.

This master's thesis presents the development of NomoPad, an iPad application that acts as the first step towards bringing back nomography in our modern technological landscape. NomoPad allows for the creation of custom nomograms for several types of equations, while guaranteeing an efficient user experience by offering features that emphasizes graphical practicalities of nomograms to a whole new degree, providing real-time interactions with nomograms such as high-level zooming and dynamic variable updates.

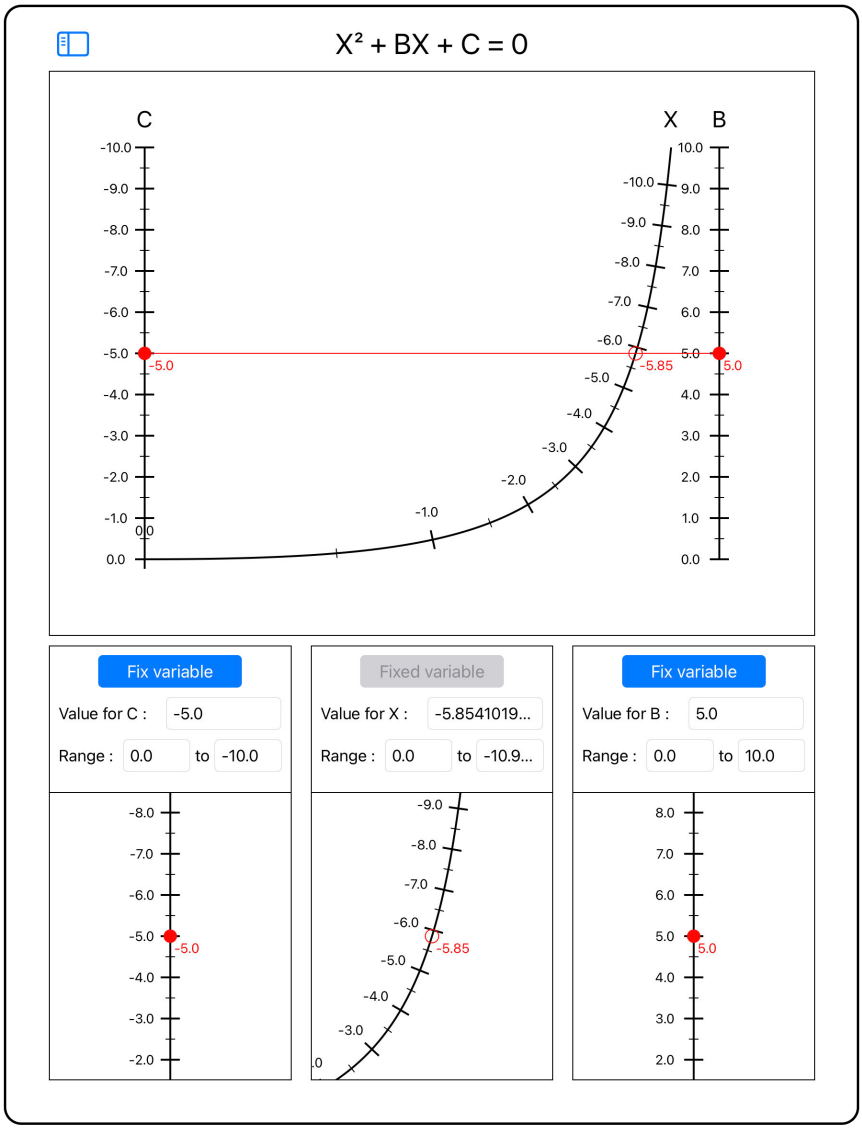

Figure 1.2: The main page of NomoPad, displaying a nomogram for an equation of the second degree

Chapter 2 explains the basic theory of nomography on which NomoPad relies to construct nomograms. Chapter 3 details the process of designing Apple apps and presents the features of NomoPad, while chapter 4 explains the implementation of these features. Chapter 5 concludes this master's thesis by presenting possible future work that could come out of it.

Chapter 2

Theory of Nomography

In the field of Nomography, the main challenge is to build nomograms for equations with 3 variables. Representing equations with 2 variables is already fully covered by analytical geometry: we just need to graph the function in a cartesian coordinate system, showing graphically the relationship between the 2 variables. Representing equations with a single variable is even more trivial, as it is done by simply drawing a horizontal line at the correct y-level.

Representing functions with more than 3 variables is, however, a complex challenge that can be done using nomography. It can be proven than any equations with more than 3 variables can be represented by the combination of several nomograms representing 3 variables but this goes beyond the scope of this master's thesis.

A nomogram representing an equation with 3 variables consists of 3 scales, one for each variable. When the index line is drawn across the diagram, it intersects with each scale and the value of the corresponding variables can be read at the intersection points. Let A, B, C be the intersections of an arbitrary straight line with the scales for a, b and c respectively, as illustrated on Figure 2.1.

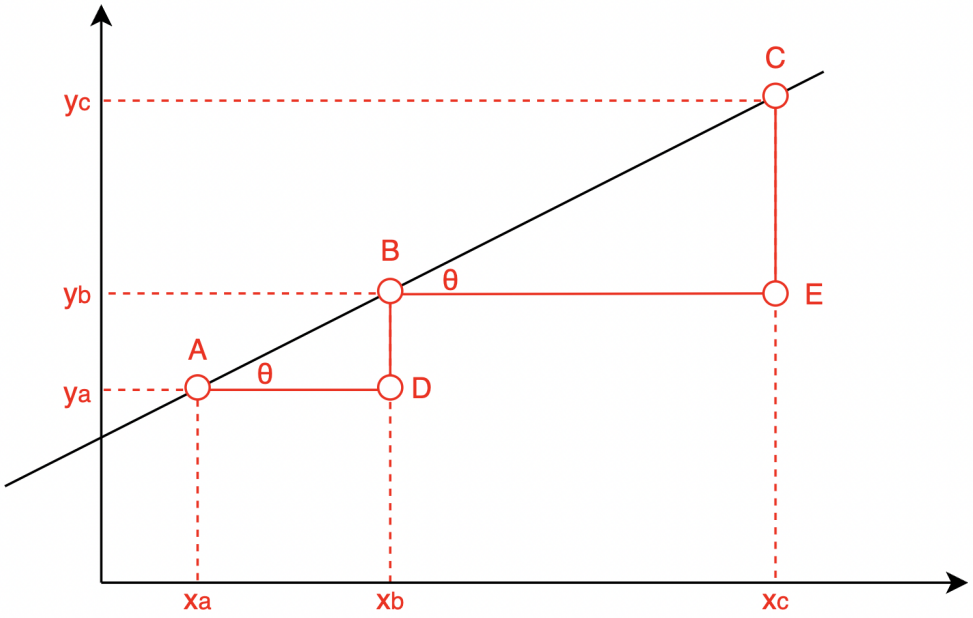

Figure 2.1

Let D = (x_c, y_a) and E = (x_c, y_b). The inclination of the line ABC corresponds to θ, the angle BAD or CBE. We have

$$tan(\theta) = \frac{BD}{AD} = \frac{y_b - y_a}{x_b - x_a} \tag{2.1}$$

and

$$tan(\theta) = \frac{CE}{BE} = \frac{y_c - y_b}{x_c - x_b} \tag{2.2}$$

Equating (2.1) and (2.2), we obtain

$$\frac{y_b - y_a}{x_b - x_a} = \frac{y_c - y_b}{x_c - x_b} \tag{2.3}$$

which after simplification gives

$$y_a x_b + y_b x_c + y_c x_a - y_a x_c - y_b x_a - y_c x_b = 0 \tag{2.4}$$

(2.4) can be written in determinant notation:

$$\begin{vmatrix} y_a & x_a & 1 \\ y_b & x_b & 1 \\ y_c & x_c & 1 \end{vmatrix} = 0 \tag{2.5}$$

If x_a and y_a, x_b and y_b, x_c and y_c are respectively only function of a, b and, c, then a nomogram can be constructed using these coordinates to form the scales of the nomogram. The nomogram is guaranteed to be correct since we showed that any 3 points a, b, and c belonging to their scales and respecting (2.5) are collinear: a line can be traced through the 3 points, it corresponds to the index line.

2.1 Nomogram for addition

In order to build a nomogram to represent the addition of 3 variables:

$$c = a + b$$

which can be rearranged to

$$a + b - c = 0 \tag{2.6}$$

In order to be represented by a nomogram, the equation (2.6) needs to be put in a determinant notation similar to (2.5), with 1's in the last column. In order to have a determinant that does not contain any products of the variables (such as ab, ac, etc.) which would violate the equation (2.6), let's put a, b and c as the first column of the matrix:

$$\begin{vmatrix} a & ? & ? \\ b & ? & ? \\ c & ? & ? \end{vmatrix} = 0 \tag{2.7}$$

The task is now to fill in the last 2 columns to get (2.6). Solutions are not unique, but can be easily found by trial-and-error on such a trivial example. Here is one of the solutions:

$$\begin{vmatrix} a & 0 & 1 \\ b & 1 & 0 \\ c & 1 & 1 \end{vmatrix} = 0 \tag{2.8}$$

By the properties of determinants, we can add the second column to the third column:

$$\begin{vmatrix} a & 0 & 1 \\ b & 1 & 1 \\ c & 1 & 2 \end{vmatrix} = 0 \tag{2.9}$$

And divide the last row by 2:

$$\begin{vmatrix} a & 0 & 1 \\ b & 1 & 1 \\ \frac{1}{2}c & \frac{1}{2} & 1 \end{vmatrix} = 0 \tag{2.10}$$

Which gives us a notation similar to (2.6) and leads to the following equations for the 3 scales a, b and c:

$$\begin{cases} y_a = a, & x_a = 0 \\ y_b = b, & x_b = 1 \\ y_c = \frac{1}{2}c, & x_c = \frac{1}{2} \end{cases}$$

By plotting and graduating the scales for a, b and c, we obtain the following nomogram which correctly represents the equation $c = a + b$:

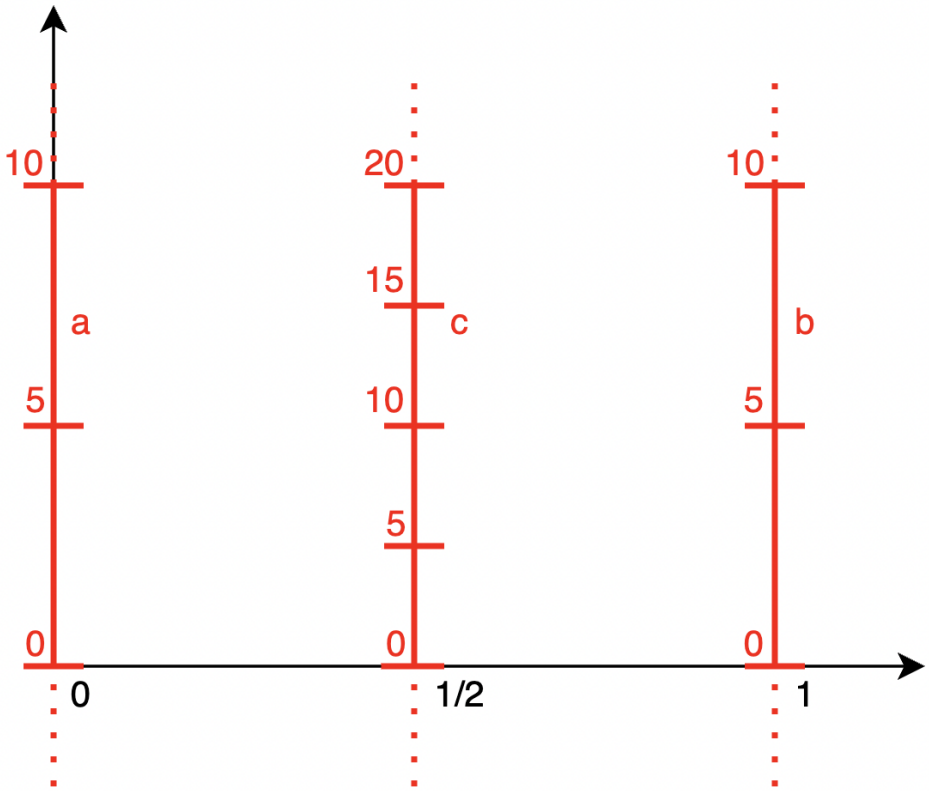

Figure 2.2: Nomogram for the equation $c = a + b$

The calculations were shown for the simple example of $c = a + b$, but it is possible to draw a nomogram for any addition of 3 functions of variables and a constant t:

$$f(c) = f(a) + f(b) + t \tag{2.11}$$

by transforming the equation back in the determinant notation (2.5). This transformation is not a trivial process and NomoPad uses a different approach to generate nomograms for additions (see 4.3.1). This approach relies on the fact that any equation of the form 2.11 can be represented using 3 parallel vertical scales as was just shown.

2.2 Nomogram for an equation of the second degree

This section explains how a nomogram for this equation:

$$a^2 + ba + c = 0 \tag{2.12}$$

can be constructed.

The approach is the same as the simple nomogram for addition: the equation (2.12) has to be transformed in the determinant notation (2.5). It can be rewritten as:

$$\begin{vmatrix} -b & 1 & 1 \\ c & 0 & 1 \\ a^2 & a & a-1 \end{vmatrix} = 0 \tag{2.13}$$

Indeed,

$$\begin{aligned} \begin{vmatrix} -b & 1 & 1 \\ c & 0 & 1 \\ a^2 & a & a-1 \end{vmatrix} &= a^2 + ca - a(-b) - (a-1)c \\ &= a^2 + ca + ab - ac + c \\ &= a^2 + ba + c \end{aligned}$$

By the properties of determinants, the last row of 2.13 can be divided by $(a-1)$:

$$\begin{vmatrix} -b & 1 & 1 \\ c & 0 & 1 \\ \frac{a^2}{a-1} & \frac{a}{a-1} & 1 \end{vmatrix} = 0$$

Which gives us a notation similar to (2.5) and leads to the following equations for the 3 scales b, c and a:

$$\begin{cases} y_b = -b, & x_b = 1 \\ y_c = c, & x_c = 0 \\ y_a = \frac{a^2}{a-1}, & x_a = \frac{a}{a-1} \end{cases}$$

The scales for b and c are vertical scales. The scale for a is represented by a curve of which the function can be computed this way:

First, in

$$y_a = \frac{a^2}{a-1}, x_a = \frac{a}{a-1} \tag{2.14}$$

we can see that:

$$y_a = x_a a \tag{2.15}$$

$$a = \frac{y_a}{x_a} \tag{2.16}$$

Thus, by substituting a in the equation of x_a in (2.14), we obtain:

$$x_a = \frac{(\frac{y_a}{x_a})}{(\frac{y_a}{x_a} - 1)}$$

$$x_a = \frac{y_a}{y_a - x_a}$$

$$y_a = x_a(y_a - x_a)$$

$$y_a = x_a y_a - x_a^2$$

$$y_a - x_a y_a = -x_a^2$$

$$y_a(1 - x_a) = -x_a^2$$

$$y_a = -\frac{x_a^2}{1 - x_a}$$

Which gives the function needed to represent the scale for a. The values along the scale are given by (2.15). Figure 2.3 shows the constructed nomogram. For simplicity, the nomogram is restricted to positive x and negative y only and the index line only intersects the a scale once, giving only one of the 2 values for a in the equation (2.12).

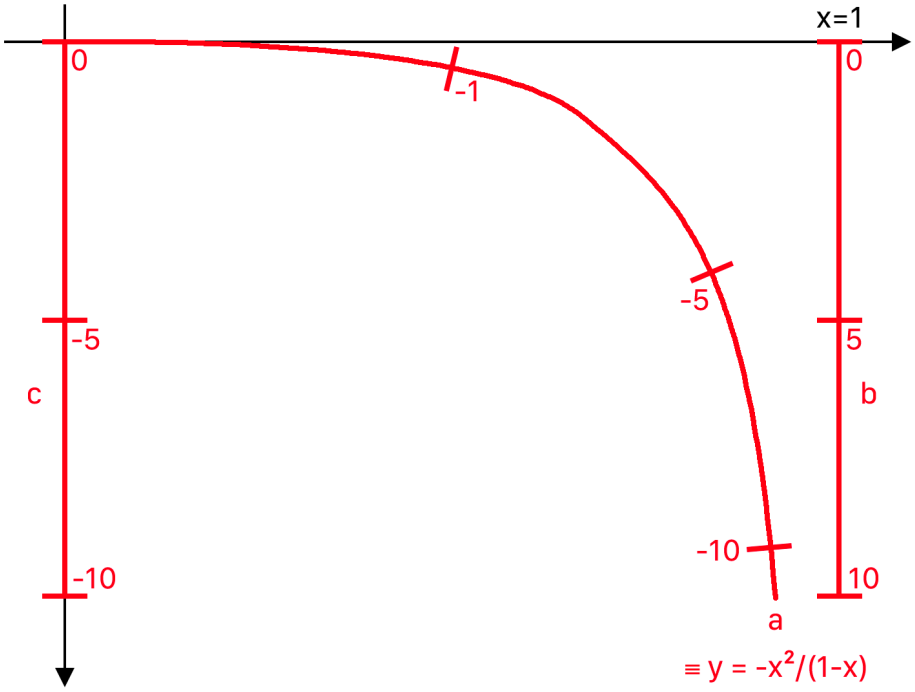

Figure 2.3: Nomogram representing the equation $a^2 + ba + c = 0$. Only displayed for positive x and negative y.

Chapter 3

NomoPad Design

This section explains how NomoPad was designed without going in the details of the implementation. The first section describes the development procedure for an iPad app using the Apple Framework. The second section goes over the structure and features of NomoPad and how it should be used.

3.1 Apple Framework

3.1.1 Development process

Developing an app for any mobile Apple device requires at least an Apple computer (Mac). The code must be written in Swift [8] or Objective-C [9] using the XCode IDE [10]. Once completed, the app can be either distributed to specific users for beta testing using TestFlight [11] or publicly uploaded to Apple's App Store after some verifications.

Swift vs. Objective-C

The programming language choice is left to personal preference. NomoPad was written in Swift since it is easier to learn than Objective-C, which is a relatively low-level language. Swift on the other hand is a high-level object-oriented language, with a syntax similar to Java. It was developed by Apple and is designed specifically to help developers create applications for the Apple ecosystem more intuitively, which is another major advantage over Objective-C.

XCode

XCode is a powerful IDE developed by Apple and usable only on Apple devices. It offers many important features such as debugging tools, version controlling and

automated app distribution. It also allows developers to test their app by allowing a local install on a connected eligible device (iPhone, iPad, etc.). In addition, XCode provides Simulators for any Apple device on which the app can be installed in case no eligible physical device is available.

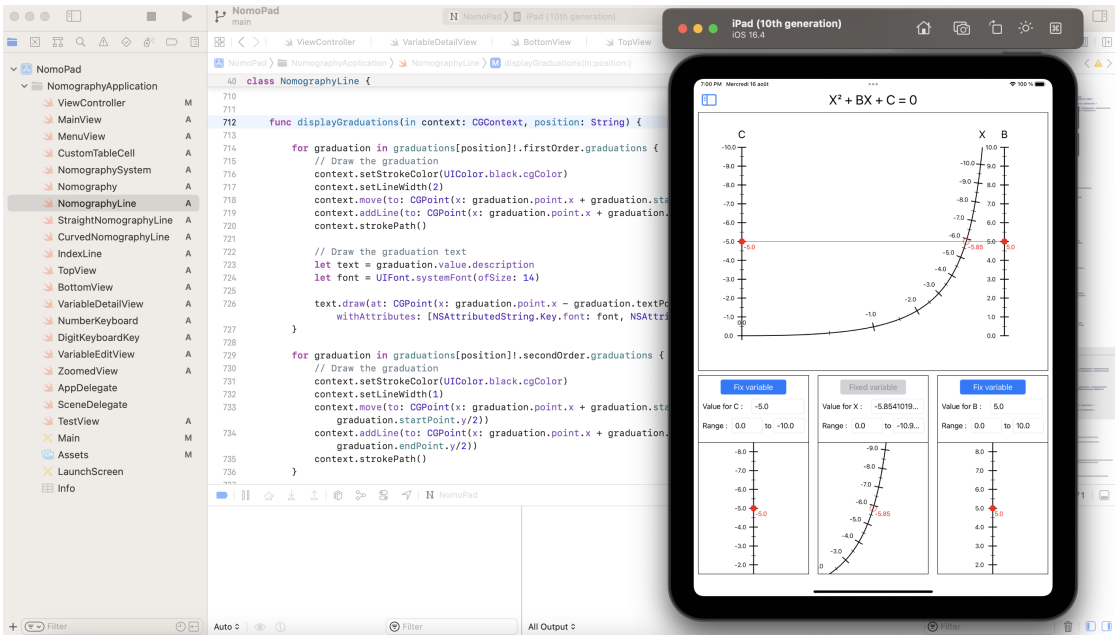

Figure 3.1: XCode interface, with the Simulator for the latest iPad running NomoPad

App distribution: App Store & TestFlight

Once the development of the app is completed, the app can be built, archived and uploaded to Apple's App Store. This process requires enrolling in the Apple Developer Program [12] and is automatically handled by XCode after providing the basic app informations: name, version and icon. After being uploaded to the App Store and being verified by Apple, the app can be either publicly released or submitted for beta testing on TestFlight.

TestFlight is the official beta testing system from Apple. It allows developers to organize private beta testing for up to 10.000 different users. These testers have to install the TestFlight app on their Apple device and are automatically notified when a new version of the app is ready for testing. Test builds of the app can be installed from the TestFlight app and expire after 90 days.

3.1.2 App design

In mobile app development, the main building blocks for creating the user interface are called Views, each corresponding to a different file (class). Views serve as graphical containers within the app's 2D coordinate system. Each View has a position and a size expressed in the app's coordinate system where the origin (0,0) is at the top-left corner. Values for position and size are expressed in pixels, the smallest discrete units on a screen. Figure 3.2 illustrates a View with a green background, a width of 400 pixels and a height of 300 pixels, positioned at (100, 100).

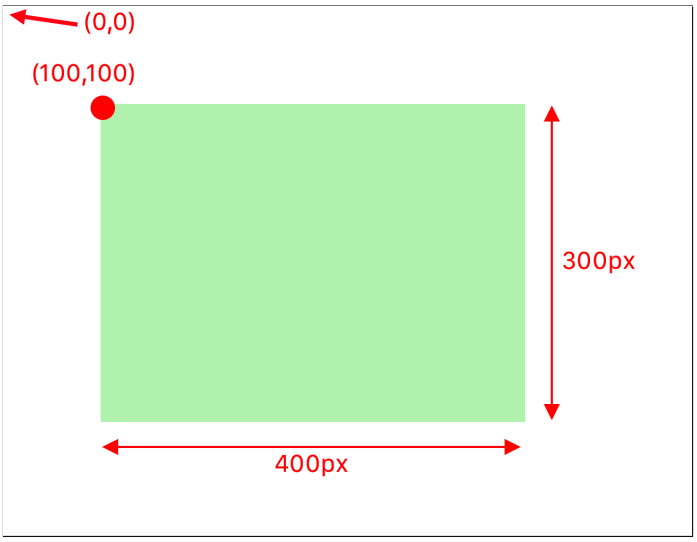

Figure 3.2

Views can contain multiple subviews, each acting as a child element. The position of a subview is defined within the coordinate system of its parent view. This means that placing a subview at coordinates (x,y) won't necessarily correspond directly to those (x,y) pixels on the screen. Instead, it aligns to (x,y) within the parent view's coordinate system, which originates from the top-left corner of the parent view. Any pixel of the subview outside of the bounds of its parent view will not be displayed on the screen. Figure 3.3 illustrates the principle of subviews.

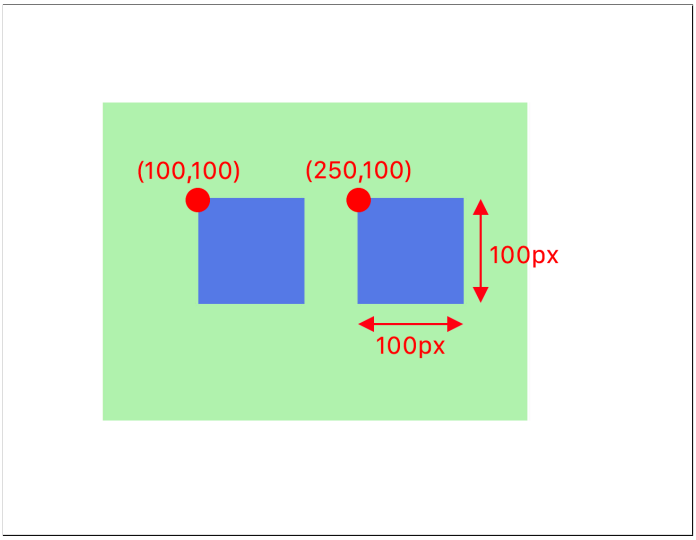

Figure 3.3

2 children blue Views of 100x100 pixels are added as subviews of the green View. Note that the subviews' positions do not correspond to the actual coordinates on the screen, but are aligned to their parent's origin as explained before.

Altough each subview's position and dimensions can be declared explicitly, a better practice is to use anchor points in order to organize subviews in a clean way. Each View has 4 main anchor points: top, bottom, left (= leading) and right (= trailing). Views can use their parent's and "siblings" Views anchor points to define their position and dimensions. Figure 3.4 shows how the 2 blue Views and be positioned using the anchor points of their parent.

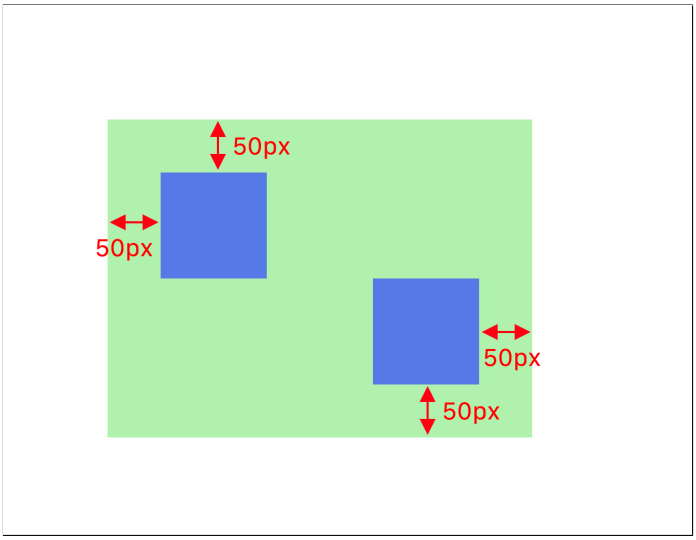

Figure 3.4

The first blue View used its parent's top anchor and leading anchor to position itself:

$$topAnchor = greenView.topAnchor + 50$$
$$leadingAnchor = greenView.leadingAnchor + 50$$

And the second blue View used its parent's bottom anchor and leading anchor:

$$bottomAnchor = greenView.bottomAnchor - 50$$
$$trailingAnchor = greenView.trailingAnchor - 50$$

With these constraints, the green View dimensions can be changed without changing the position of the subviews: the first blue View will stay at the top left and the second will stay at the bottom right.

3.2 NomoPad

3.2.1 App structure

NomoPad makes great use of the View-subview hierarchy, coordinates system and constraints system using anchor points decribed before in order to lay out all the different parts of the app on the screen correctly.

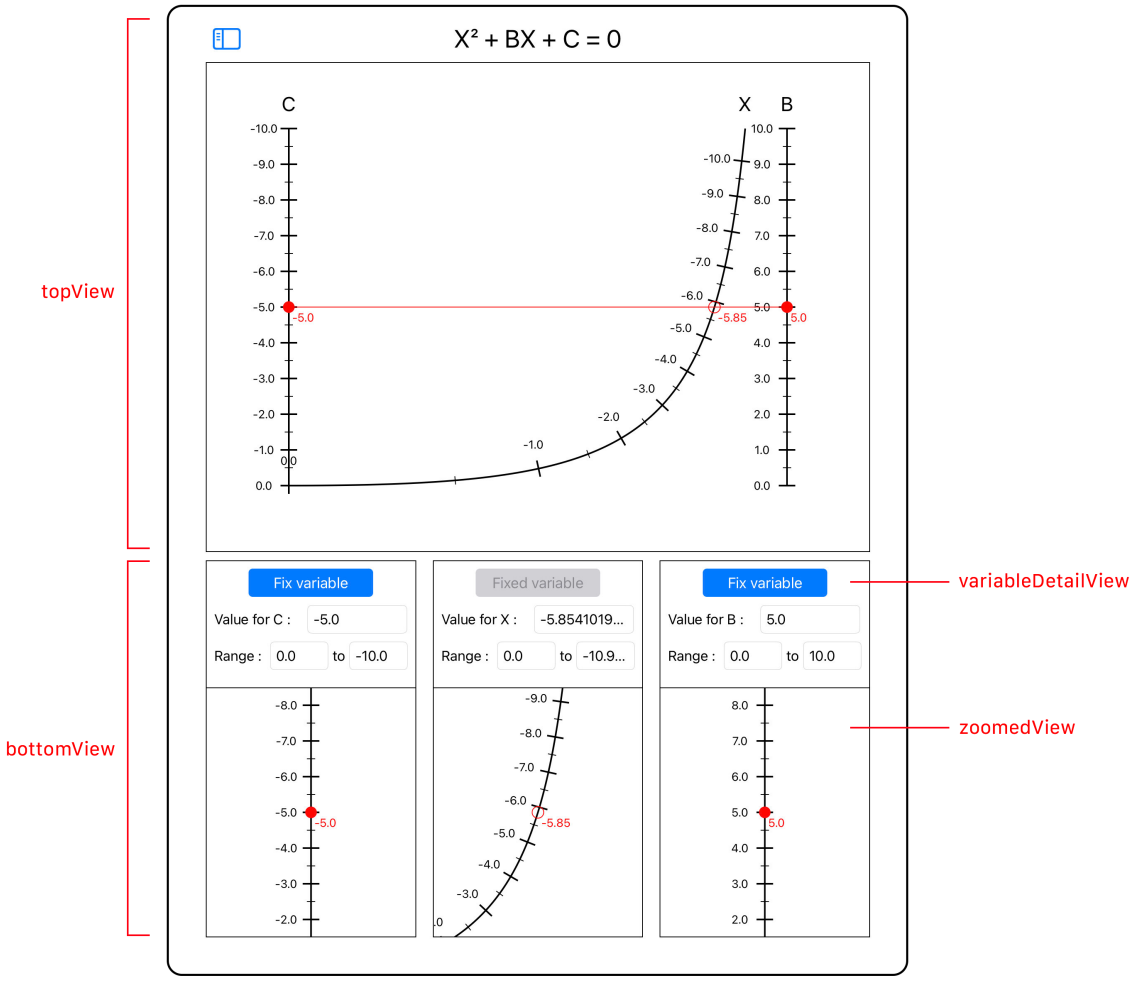

Figure 3.5: Structure of the main page of NomoPad

The app is composed of 3 main Views:

- **topView**: Taking the top half of the main screen, the topView displays the selected nomogram as well as the equation it represents.

- **bottomView**: Taking the bottom half of the main screen, this View contains 3 subviews for each variable represented by the selected nomogram. Each subview is itself composed of 2 subviews:
  - A **variableDetailView** that gives detailed information about the variable.
  - A **zoomedView** that displays a zoomed version of the scale representing the variable, with its current value always being at the center of the View.
- **menuView**: A side panel that can be opened and closed, it allows to browse through the different nomograms and select which nomogram to display.

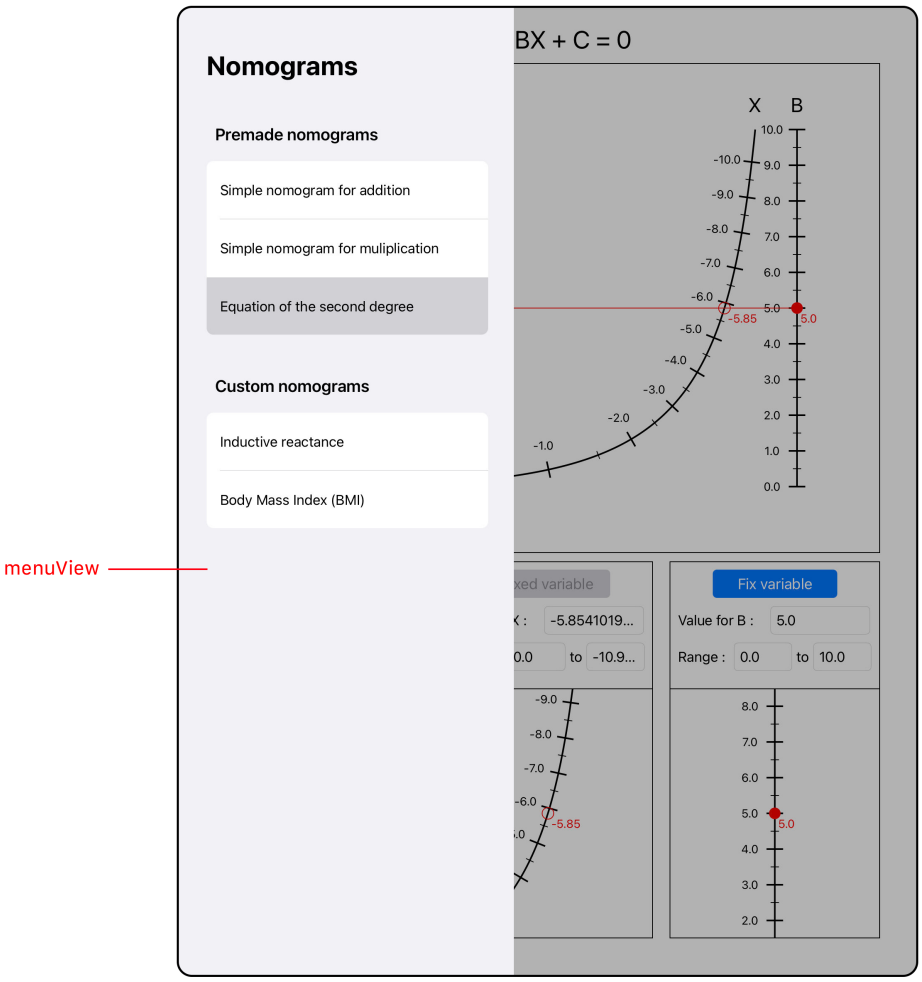

Figure 3.6: The menuView of NomoPad

In NomoPad, each View is actually a UIView, a specific type of View that has a **draw** method. This function is key for displaying nomograms efficiently. It allows to draw lines, curves and text on the screen by using the 2D coordinates system of the UIViews.

3.2.2 App features & Usage guide

Changing variable values

NomoPad allows users to change the values of the variables represented by the 3 scales of the nomogram. In order to change the value of a variable while maintaining the relationship linking the 3 variables, the value of one of the 2 other variables has to be fixed. This way, the value of the third variable can be computed according to the change in the variable value. A fixed variable is represented on the nomogram by an empty red dot, while an unfixed variable is represented by a filled red dot. On startup, one of the 2 input variables will be fixed, typically the one on the right of the screen, since the typical action a user will want to do is update one of the inputs to see the result on the output.
A variable's value can be fixed by using the "Fix variable" button in the variableDetailView, or by a prolonged touch on the value of the variable along the corresponding scale in the topView. Fixing the value of a variable automatically unfixes the 2 other variable values.

The value of a variable can be changed in 2 ways:

- The new value can be explicitly inputted by the user in the corresponding the text field in the variableDetailView
- The user can drag the filled dot of the variable value in the topView. The red dot will follow the user's finger and the value of the variable will be updated accordingly.

Trying to change the value of a fixed input variable will immediately unfix it and fix the other input variable.

Changing variable ranges

The user can change the range a variable using the appropriate text fields in the variableDetailView. The range of the output variable cannot be changed since it depends on the range of the 2 input variables. Once the range of a variable is updated, the whole nomogram is reconstructed to account for the change.

Zooming

Zooming can be done both on the main display of the whole nomogram (topView) and the 3 zoomed views of the scales. On the topView, users can pinch to perform a zoom/dezoom on any part of the nomogram. For better user experience, the zoom is centered on the positions between the users's fingers. As zoom or dezoom is performed, graduations on the scale are dynamically updated with an unnoticeable delay, ensuring a smooth experience. Graduation values are updated when required: when zooming, as distance between the graduations become too large, new graduations appear for better precision. Inversely, when dezooming, as graduations become too close to each other, some graduations disappear and the most important ones are kept. In NomoPad, graduations are built to be monotonic:

- When zooming and adding new graduations, all the old graduations are still present.
- When dezooming and old graduations are removed, no new graduations are created.

This ensures a coherent user experience. Figures 3.7 and 3.8 show the effect of zooming on the curved scale representing the variable X. As we can see, the curved line becomes almost flat after the local zoom and the graduations have been updated for more precision. The displayed variable value at the intersection with the index line shows more decimal points to reflect better precision as well.

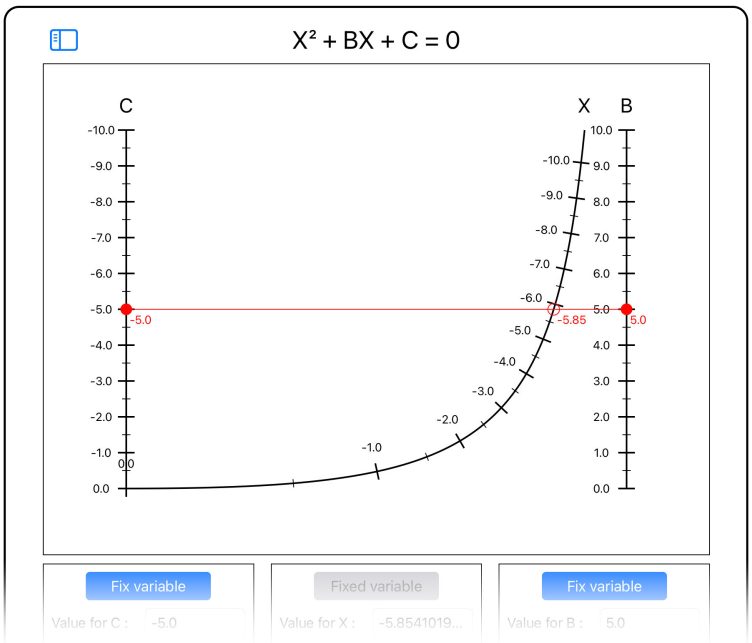

Figure 3.7: Original scale

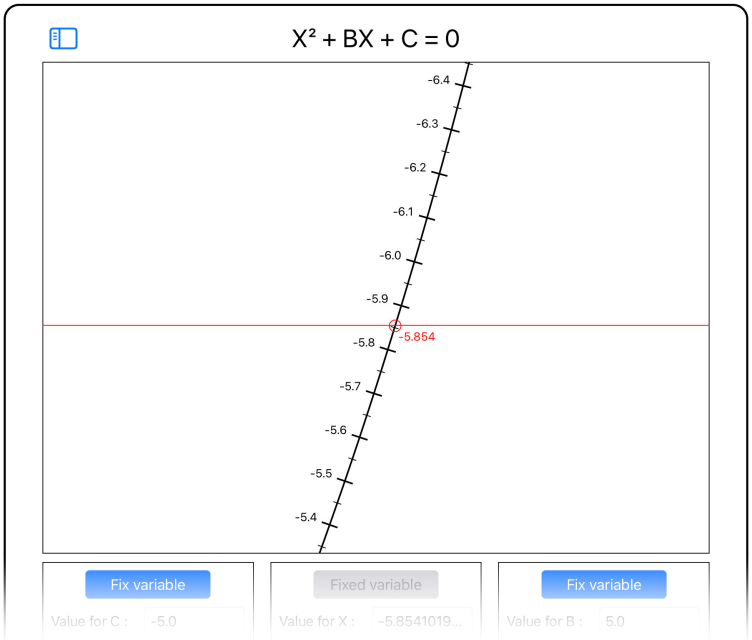

Figure 3.8: After zooming on the X scale

The purpose of the 3 zoomed Views is to focus zooming on the intersection point between each scale and the index line. The value of the variable on the scale is always displayed at the center of each View, therefore zoom cannot be applied at any location like on topView: the zoom origin is always at the center of the zoomed Views. Since the 3 zoomed Views are independent of each other, zoomed can also be applied independently on any View. Fig 3.9 shows the 3 zoomed Views each on being at different zoom levels.

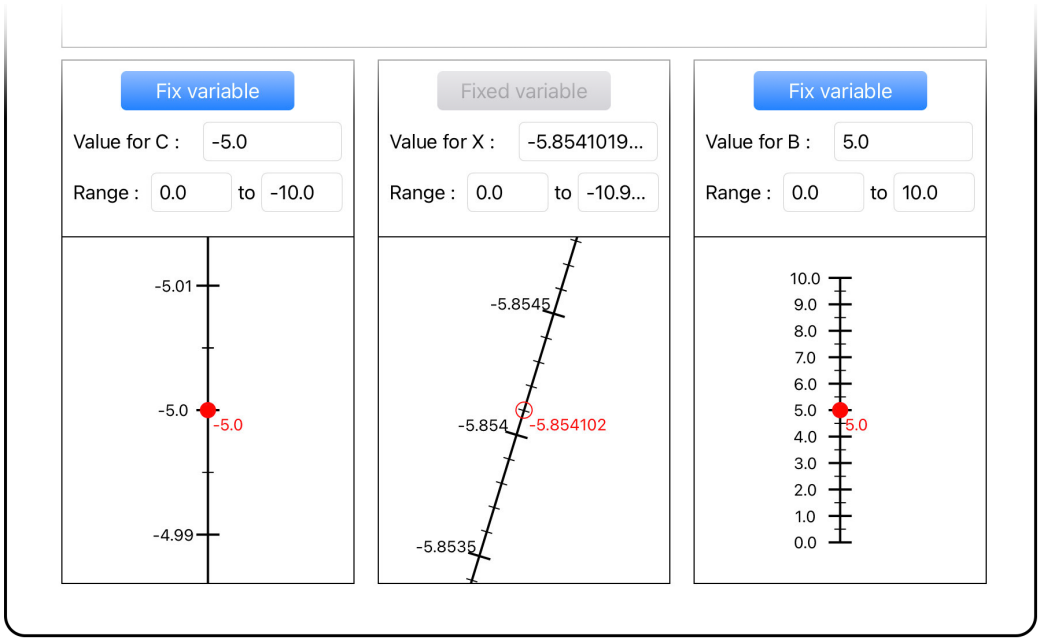

Figure 3.9: The 3 zoomed views at different zoom levels

Panning

NomoPad supports panning on the topView: users can drag their finger and the nomogram will be translated across the screen, following the movement of their finger. Note that the dragging the finger over the variable value along the scale will always have priority over the translation of the nomogram, if the value of the variable can be changed.

Chapter 4

NomoPad Implementation

This chapter goes over the details of implementation of the main features of the app:

- **Scales**: this section explains how each type of scales used for representing the different variables of the nomogram are generated
- **Graduations**: this section goes over the most important features of scales: their graduations, how they are generated and which values are chosen to be displayed
- **Nomograms**: in this section, the construction of each type of nomogram using the previous building blocks (graduated scales) is detailed
- **User interaction**: this section details how zooming and panning is implemented and optimized for better performances
- **View syncing**: this final section explains how the topView and the 3 zoomedViews are synchronized to ensure a coherent user experience

4.1 Scales

The base building blocks for constructing a nomogram are the scales representing the variables. In NomoPad, two different types of scales are implemented: **straight scales** and **curved scales**. Here is the list of the common features defining a scale of any type:

- **Variable name**: The name of the variable represented by the scale.
- **Variable function**: The function (= factor & exponent) of the variable represented by the scale.

- **Variable range**: The start & end values of the variable value represented by the scale.
- **Graduations**: The list of the first & second order graduations displayed along the scale.
- **Start/end position**: The (x,y) coordinates on the screen of the start & end position of the scale.

Scales have a common important function that is used to generate their graduations (4.2), build the nomogram (4.3), handling user interaction (4.4) and syncing the displays (4.5):

- **getPoint**: Computes the (x,y) coordinates on the screen of a given value of the variable represented by the scale. These coordinates are always inside the scale.

4.1.1 Straight scale

A straight scale is always represented as a vertical line, starting at the bottom of the main view screen and ending at the top. A straight scale can be either **linear** or **logarithmic**, depending on the variable it represents. Figure 4.1 illustrates these 2 types of scales using the same range. For instance, constructing a nomogram for the equation

$$log_{10}(C) = 2A - B$$

will require a logarithmic scale to represent variable C, and 2 linear scales for variables A and B.

Linear scale

On a linear scale, the graduations are spaced evenly. The implementation of the **getPoint** function for a linear straight scale is detailed in Algorithm 1.

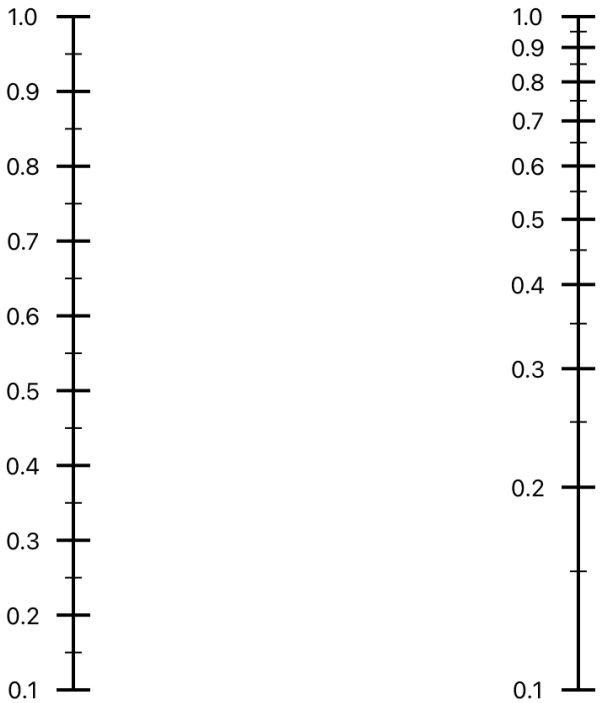

Figure 4.1: NomoPad: Linear scale (left) vs. logarithmic scale (right)

Algorithm 1

1: **function** GETPOINTLINEAR($scale : Scale, value : Double$)
2: $startY \leftarrow scale.startPosition.y$ ▷ The y-coordinate of the start point
3: $endY \leftarrow scale.endPosition.y$ ▷ The y-coordinate of the end point
4: $startValue \leftarrow scale.startValue$
5: $endValue \leftarrow scale.endValue$
6: $relativeValue \leftarrow \dfrac{value - startValue}{endValue - startValue}$
7: $y \leftarrow startY + relativeValue * (endY - startY)$
8: $x \leftarrow scale.startPosition.x$
9: **return** (x, y)

First, the **relative value** of the given value is computed using the bounds of the scale. It expresses how far along the range of the scale the given value is. Since the scale is linear, this relative value can be directly used to compute the y-position. The x-position is equal to the x-position of the scale, as it is always a vertical line.

Logarithmic scale

On a logarithmic scale, the graduations are not spaced evenly, thus the simple **getPoint** function of linear scales using the relative value to compute the position of a given value cannot be used. A different approached is detailed in Algorithm 2.

Algorithm 2

1: **function** GETPOINTLOG($scale : Scale, value : Double$)
2: $startPos \leftarrow scale.startPosition$
3: $endPos \leftarrow scale.endPosition$
4: $logStartValue \leftarrow log_{10}(scale.startValue)$
5: $logEndValue \leftarrow log_{10}(scale.endValue)$
6: $logValue \leftarrow log_{10}(value)$
7: $newScale \leftarrow Scale(startPos, endPos, logStartValue, logEndValue)$
8: $(x, y) \leftarrow getPointLinear(newScale, logValue)$
9: **return** (x, y)

This time, the given value cannot be directly positioned on the scale using its relative value since the scale is not linear. A new linear scale is created at the same position as the logarithmic scale, but with logarithms of its bounds. The logarithm of the given value is placed linearly on this new scale, giving the correct position on the initial scale. Figure 4.2 illustrates the algorithm on a concrete example of positioning the value 0.5 on a logarithmic scale going from 0.1 to 1.0.

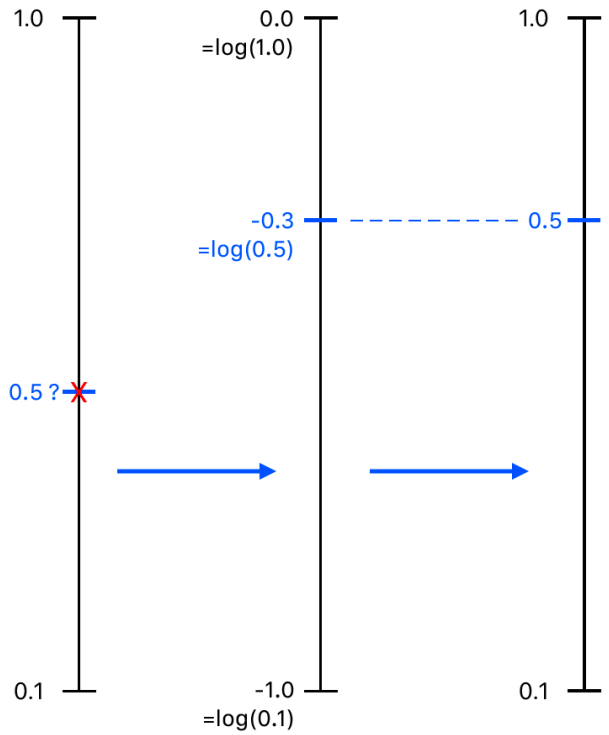

Figure 4.2: Illustration of the **getPoint** function for a logarithmic scale

4.1.2 Curved scale

In order to draw the curved line representing the curved scale, there are 2 options:

- Simulating a curved line using multiple straight lines. This approach is useful to represent any function y = f(x), assigning (x,y) coordinates in the 2D coordinate system of the screen. However, in order to draw a smooth curved line, the amount of straight lines required is very high, which is computationally expensive especially since NomoPad supports zooming and panning: applying these transformations to all the straight lines everytime the user interacts with the nomogram would make the app very slow.

- Using cubic Bézier curves. This approach allows for drawing smooth curves using only 4 parameters: the start and end points of the curve alongside 2 control points used to shape the curve. The disadvantage of this method is that representing complex curved functions using Bézier curves is not a trivial task: that may require to combine multiple Bézier curves and each control point has to be estimated correctly in order to fit as best as possible to the true curve.

The Bézier curve approach was chosen to represent curved scales in NomoPad due to its remarkable ability to create fluid and smooth curves with minimal input, making the implementation way simpler and more intuitive. Section 4.3.3 explains how the control points of the Bézier curve used in the nomogram are computed. The **getPoint** function of the curved scale is also detailed in this section since it depends on the other scales of the nomogram. Figure 4.3 shows an example of a cubic Bézier curve, showing the purpose of the 2 control points P1 and P2 for shaping the curve.

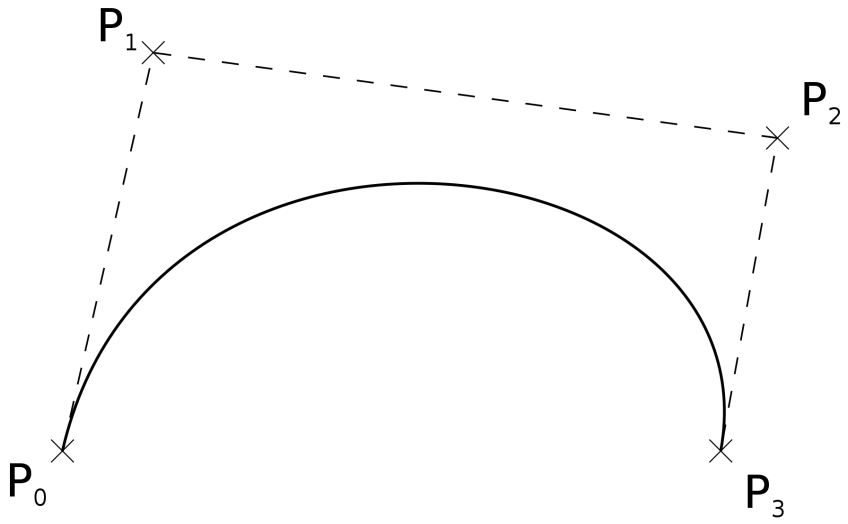

Figure 4.3: Example of a cubic Bézier curve. P1 and P2 are the 2 control points. [13]

4.2 Graduations

The next step to building a full nomogram is to graduate the scales. Without proper graduations, the different variables are not correctly represented and the graphical intuition offered by nomograms vanishes. Graduating the scales is a 2-step process:

1. Choose which graduations to display for each scale
2. Position these graduations correctly on the scales

The second step is a trivial task: as it was detailed in the previous sections, each type of scale has its own **getPoint** function that returns the position of a given value of the variable represented along that scale. By invoking this function for each graduation value chosen to be displayed, we can retrieve their correct position.

Choosing which graduations to display for each scale, however, is a complex problem:

- Graduation values must be coherent. For example, a scale ranging from 1 to 5 could be graduated with values 1.5, 2.5, 3.5 and 4.5, but a more natural way is of course to use graduations 1, 2, 3, 4 and 5. **(Coherence)**
- There must be enough graduations to allow for fast variable value retrieval, be not too many to avoid overwhelming the display. **(Right amount)**
- Since NomoPad supports zooming and dezooming, graduations values are not immutable: they must evolve based on the current zoom level to ensure a correct display in any situation. Moreover, to ensure a decent user experience, graduations values have to be monotonic: when the user zooms, new graduations should appear as needed while still keeping the same previous graduations. Inversely, when dezooming, no new graduation value should appear: only irrelevant graduations should disappear. **(Monotonicity)**

The graduation system of NomoPad satisfies all these conditions. It is inspired by classic graduated rulers, such as the one shown in Figure 4.4.

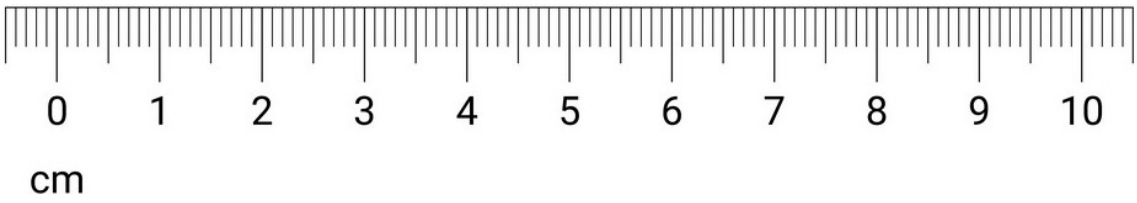

Figure 4.4: Classic graduated ruler [14]

As we can see, graduations on classic rulers are organized in 3 different orders, from largest to smallest:

- **First order**: These graduations are multiples of $1 = \mathbf{10^0}$. Their values are displayed next to them.
- **Second order**: These graduations are halfway between the first order graduations, they are multiples of $0.5 = \frac{\mathbf{1}}{\mathbf{2}} \times \mathbf{10^0}$.
- **Third order**: These are the smallest graduations, multiples of $0.1 = \mathbf{10^{-1}}$.

If we were to add a fourth order, the graduations would be multiples of $0.05 = \frac{\mathbf{1}}{\mathbf{2}} \times \mathbf{10^{-1}}$. A pattern emerges from this:

- Graduations are always either multiples of $\mathbf{10^N}$ or $\frac{\mathbf{1}}{\mathbf{2}} \times \mathbf{10^N}$ ($N \in \mathbb{Z}$), alternating from one order to the next.
- Going to the next order (= "zooming") from an order where graduations are multiples of $\frac{\mathbf{1}}{\mathbf{2}} \times \mathbf{10^N}$ decreases N by 1: the next order graduations are multiples of $\mathbf{10^{N-1}}$
- Going to the previous order (= "dezooming") from an order where graduations are multiples of $\mathbf{10^N}$ increases N by 1: the previous order graduations are multiples of $\frac{\mathbf{1}}{\mathbf{2}} \times \mathbf{10^{N+1}}$

NomoPad uses a similar graduation system, except that each scale is graduated only with graduations of **first order** and **second order**. Thus, at any zoom level, graduations can either be defined by:

$$\text{Graduations} = \begin{cases} \textbf{First order} \in \text{multiples of } 10^N \\ \textbf{Second order} \in \text{multiples of } \frac{1}{2} \times 10^N \end{cases} \tag{4.1}$$

or:

$$\text{Graduations} = \begin{cases} \textbf{First order} \in \text{multiples of } \frac{1}{2} \times 10^N \\ \textbf{Second order} \in \text{multiples of } 10^{N-1} \end{cases} \tag{4.2}$$

while keeping the graduations inside the bounds of the scale.

Figure 4.5 illustrates these 2 possible situations:

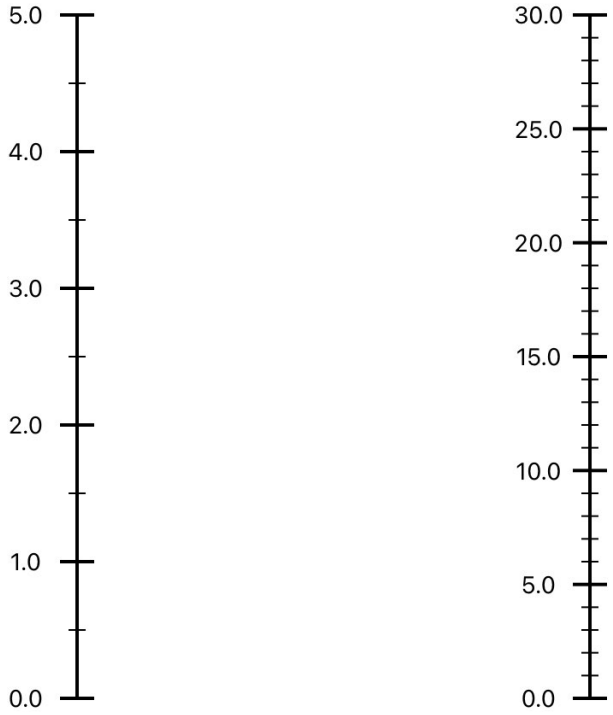

Figure 4.5: NomoPad: illustration of different zoom levels

On the left scale graduated from 0 to 5:

$$\text{Graduations} = \begin{cases} \textbf{First order} \in [0, 1, 2, ...] = \text{multiples of } 10^0 \\ \textbf{Second order} \in [0, 0.5, 1, 1.5, ...] = \text{multiples of } \frac{1}{2} \times 10^0 \end{cases}$$

On the right scale graduated from 0 to 30:

$$\text{Graduations} = \begin{cases} \textbf{First order} \in [0, 5, 10, ...] = \text{multiples of } \frac{1}{2} \times 10^1 \\ \textbf{Second order} \in [0, 1, 2, ...] = \text{multiples of } 10^0 \end{cases}$$

Let's see point by point how this graduation system satisfies the conditions stated before:

Coherence

Since the graduation system of NomoPad is directly inspired by real-life graduated rulers, we can safely say the graduations are coherent.

Right amount

For any scale in NomoPad, the first order graduations are initialized based on the bounds of the scale. More specifically, first order graduations are initialized to multiples of 10^N, with

$$N = \frac{log(|upperBound - lowerBound|)}{10}$$

For example, a scale going from 0 to 1000 will have first order graduations multiples of $10^{log(100)} = 10^2 = [0, 100, 200, ...]$. This ensures that first order are initialized to a reasonable amount. By (4.1), second order graduations are initialized to multiples of $\frac{1}{2} \times 10^2 = [0, 50, 100, 150, ...]$. Therefore, both first and second order graduations are initialized to a right amount.

Monotonicity

To prove the monotonicity of this graduation system, we will use these properties:

$$k \times 10^N \subset k \times 10^M \tag{4.3}$$

$$k \times \frac{1}{2} \times 10^N \subset k \times \frac{1}{2} \times 10^M \tag{4.4}$$

$$\forall k, M, N \in \mathbb{Z}, \text{with } M \leq N$$

When the zoom level changes, the first order and second order graduations are recomputed w.r.t. (4.1) and (4.2). Monotonicity is proven for increasing zoom level. The proof for decreasing zoom level being analogous, it is not detailed.

Increasing zoom level: The old second order graduations become the new first order graduations. Therefore, [old second order] $\subset$ [new first order].

- In the case of (4.1), old first order graduations $\in$ multiples of 10^N. The new graduations of second order are assigned to multiples of 10^{N-1}. Therefore, by (4.3), [old first order] $\subset$ [new second order].
- In the case of (4.2), old first order graduations $\in$ multiples of $\frac{1}{2} \times 10^N$. The new graduations of second order are assigned to multiples of $\frac{1}{2} \times 10^{N-1}$. Therefore, by (4.4), [old first order] $\subset$ [new second order].

In every case, we proved [old second order] $\subset$ [new first order] and [old first order] $\subset$ [new second order]. Therefore, we can say [old graduations] $\subset$ [new graduations], proving monotonicity.

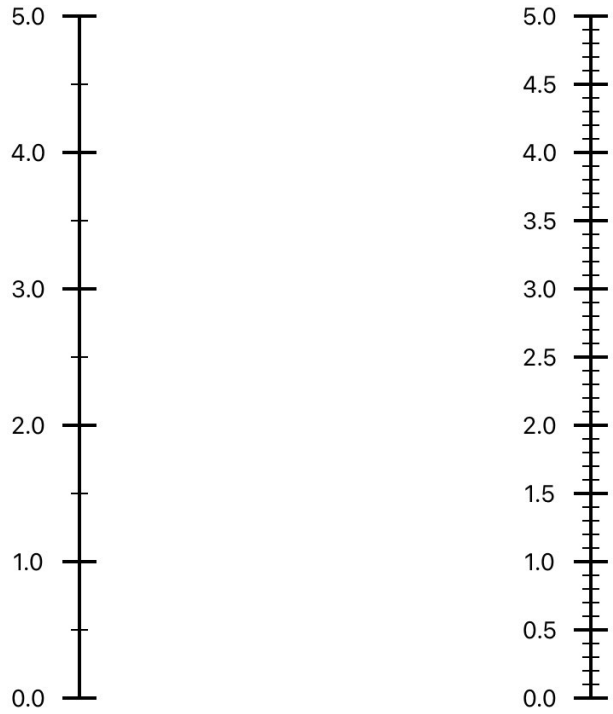

Figure 4.6: NomoPad: illustration of the monotonicity of the graduations

Figure 4.6 shows this monotonicity property intuitively. When increasing the zoom level from the left scale to the right scale, it can be seen that all previous graduations are kept after the zoom and more graduations are added. Inversely, when decreasing the zoom level, no new graduation are added, only irrelevant graduations are removed. This ensures monotonicity of the graduations.

4.3 Nomograms

Now that we explained the implementation of correctly graduated scales of any type, it is time to build actual nomograms using these scales. This section explains how the 3 nomogram types supported by NomoPad are created:

- **Nomogram for addition**: $f(c) = f(a) + f(b) + t$, t being a constant
- **Nomogram for multiplication**: $f(c) = f(a) \times f(b)$
- **Nomogram for this equation of the second degree**: $x^2 + bx + c = 0$

For each nomogram, the process of creating the 3 scales that define it as well as their position and values is detailed.

4.3.1 Nomograms for addition

As seen in the theory, an addition of the type $f(c) = f(a) + f(b) + t$ can be represented using 3 parallel vertical scales. We will use this assumption to create nomograms for addition with a more intuitive method than the one detailed in the theory. Since 3 parallel vertical scales can always represent an addition, let's first place the 2 scales representing the variables for which the range is given (a and b) and compute the position (= x-coordinate) of the c scale accordingly.

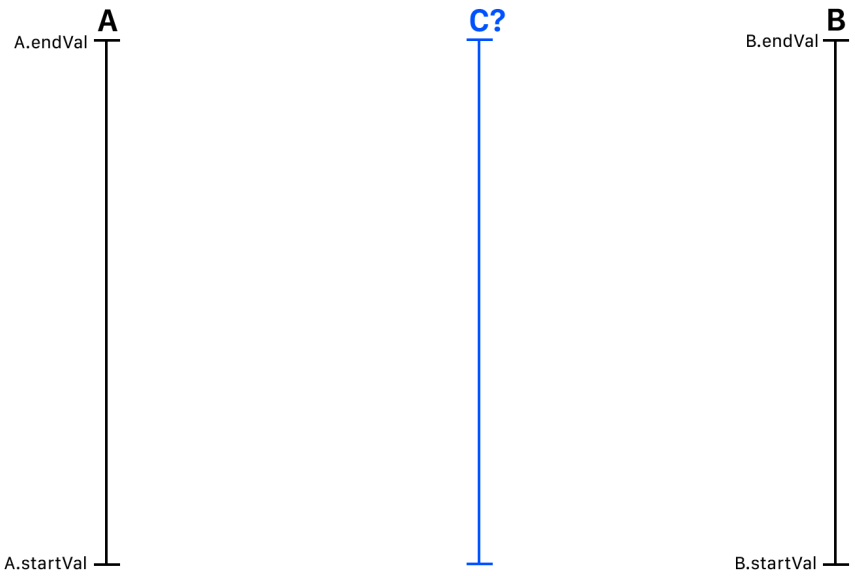

Figure 4.7

In order to determine the position of C, we need to find two different pairs of values for A and B (a_1, b_1), (a_2, a_2) leading to the same value for C, mathematically:

$$f(a_1) + f(b_1) + t = f(c) \tag{4.5}$$

$$f(a_2) + f(b_2) + t = f(c) \tag{4.6}$$

This way, the x-coordinate of the C scale will be given by the x-coordinate of the intersection between the 2 distinct lines (a_1, b_1) and (a_2, b_2).

Finding the pairs (a_1, b_1), (a_2, a_2) that solve (4.5) and (4.6) is equivalent to solving this equation:

$$f(a_1) + f(b_1) = f(a_2) + f(b_2) \tag{4.7}$$

Let us assign these values for a_1, b_1 and a_2:

$$a_1 = A.endValue$$

$$a_1 = B.startValue$$

$$a_2 = A.startValue$$

Solving (4.7) for b2 will give the two distinct pairs $(A.endVal, B.startVal)$, $(A.startVal, b_2)$:

$$\begin{aligned} f(A.endVal) + f(B.startVal) &= f(A.startVal) + f(b_2) \\ f(b_2) &= f(A.endVal) + f(B.startVal) - f(A.startVal) \\ b_2 &= f_B^{-1}(f(A.endVal) + f(B.startVal) - f(A.startVal)) \end{aligned}$$

Finally, the x-coordinate of the C scale is given by:

$$[(A.endVal, B.startVal) \cap (A.startVal, b_2)].x$$

Since the 3 scales for A, B and C start and end at the same y-coordinate, the start and end points of the C scale can be obtained using x-coordinate previously computed and the y-coordinates of the start and end points of A and B.

The last element needed to build the C scale are its start and end values. It is obtained using the start and end values of A and B:

$$f(C.startVal) = f(A.startVal) + f(B.startVal) + t$$
$$C.startVal = f_C^{-1}(f(A.startVal) + f(B.startVal) + t)$$

The computation of its end value is analogous. Using its start and end positions, we can build the C scale and graduate it from its start value to its end value, which completes the construction of the nomogram.

As a practical example, let's construct a nomogram for the simple addition:

$$C = \frac{1}{2}A + \frac{1}{2}B$$

with

$$0 \leq A \leq 5, 0 \leq B \leq 10$$

We first compute the value of b_2 :

$$\begin{aligned} b_2 &= f_B^{-1}(f(A.endVal) + f(B.startVal) - f(A.startVal)) \\ &= f_B^{-1}(\frac{1}{2} \times 5 + 0 - \frac{1}{2} \times 0) \\ &= f_B^{-1}(2.5) \\ &= 2 \times 2.5 \\ &= 5 \end{aligned}$$

From b_2, the intersection point can be computed, from which the start and end points of the C scale can be obtained. Let's compute the start and end values of C

$$\begin{aligned} C.endVal &= f_C^{-1}(f(A.endVal) + f(B.endVal) + t) \\ &= f_C^{-1}(\frac{1}{2} \times 5 + \frac{1}{2} \times 10) \\ &= f_C^{-1}(7.5) \\ &= 7.5 \end{aligned}$$

An analogous process gives $C.startVal = 0$

The construction of the C scale is illustrated on Figure 4.8:

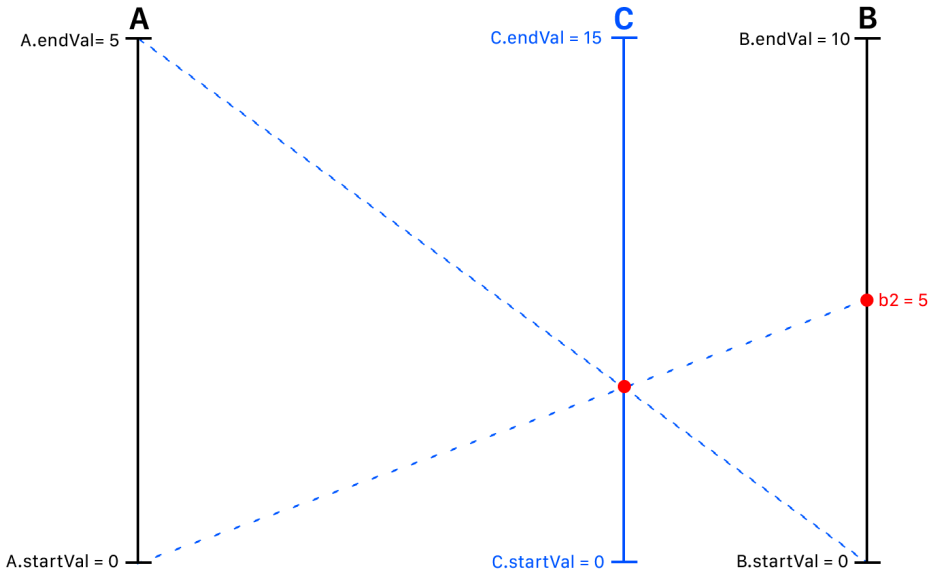

Figure 4.8: Computation of the x-position and the bounds of the C scale

And here is the full nomogram built in NomoPad:

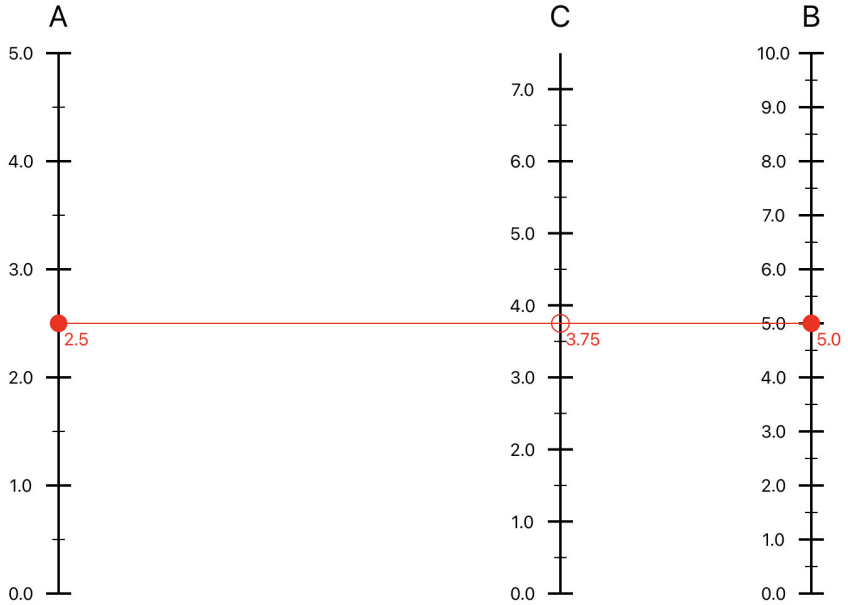

Figure 4.9

4.3.2 Nomograms for multiplication

As seen in the theory, a multiplication of the type $f(c) = f(a) \times f(b)$ can also be represented using 3 parallel vertical scales. As in the addition nomogram, the ranges of the two right-hand scales representing the variable A and B are given. The start and end positions of the A and B scales are also known. Computing the position of the C scale becomes trivial after applying the log operator on both sides of the equation:

$$log_{10}(f(c)) = log_{10}(f(a) \times f(b))$$
$$log_{10}(f(c)) = log_{10}(f(a)) + log_{10}(f(b))$$

Using a nomogram for addition with f(c) = $log_{10}(f(c))$, f(a) = $log_{10}(f(a))$ and f(b) = $log_{10}(f(b))$, we can compute the position of the c scale for the multiplication equation. While their positions remains unchanged, the values of each scale must be raised to the power of 10 in order to correctly represent the original equation. Figure 4.10 shows a nomogram for a simple multiplication $c = ab$.

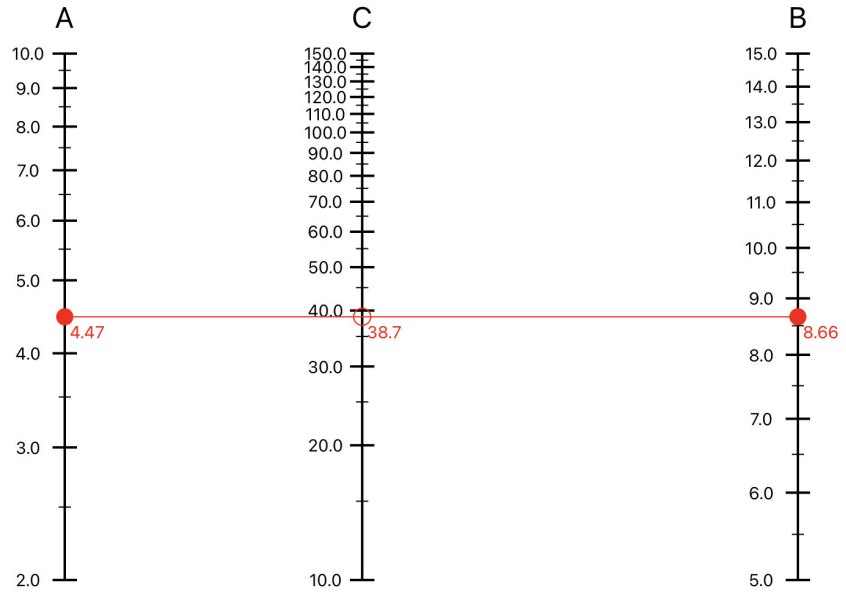

Figure 4.10: NomoPad: a nomogram representing the equation $c = ab$

Note that because we passed logarithmic functions to the addition nomogram, all the scales representing the multiplication nomogram are logarithmic.

4.3.3 Nomogram for an equation of the second degree

As seen in the theory, transforming an equation of the second degree into the correct determinant form and from it, the computation of the function used to represent the curved scale are both complex tasks. Due to a lack of time, NomoPad only supports the very equation that has been dealt with in the theory:

$$a^2 + ba + c = 0$$

The nomogram constructed in NomoPad for this equation acts ask a "proof-of-concept", showing that a nomogram with a curved scale can be built and displayed correctly, as well as allowing the same user interaction as any nomogram for addition or multiplication.

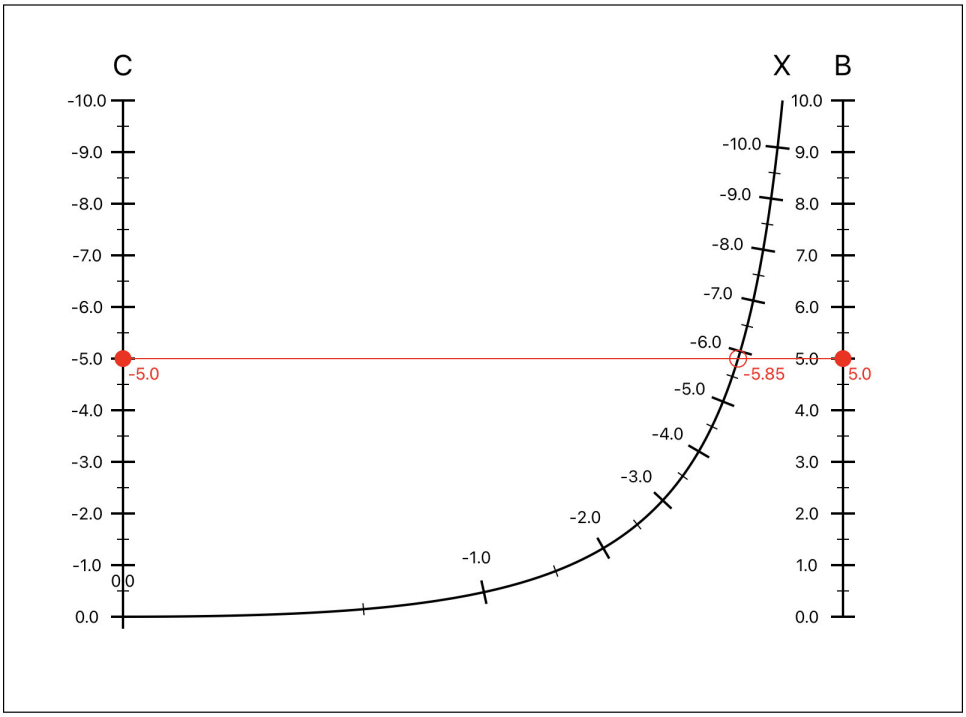

Figure 4.11: NomoPad: nomogram for the equation $x^2 + bx + c = 0$

Note that the variable a has been replaced by the variable X in NomoPad. In addition, the y-scale has been reversed for a clearer display. Thus, the nomogram is vertically flipped compared to the one seen in the theory.

The construction of the 2 vertical scales representing the variables B anc C is trivial. The main task consists in building the curved scale for the X variable,

which is done by computing the correct Bézier control points that control the shape of the curve. We need to build a curve that fits as close as possible to the true curve defined by the function:

$$y_a = -\frac{x_a^2}{1 - x_a} \tag{4.8}$$

The Bézier control points can be estimated using this approach [15]:

1. Draw the tangent lines to the start and end point of the curve. Since the start and end point of the Bézier curve as equivalent to those of the true curve, their tangent lines can be drawn using the derivative of (4.8)
2. Draw a line tangent to the true curve that is parallel to the line connecting the start and end points of the curve. Given derivative of the equation (), the tangent line corresponding to a given slope can easily be drawn. The slope is computed using the start and end points of the curve.
3. Compute P_r and P_s, the two intersection points between the first 2 tangent lines drawn in step 1 and the tangent line computed in step 2.
4. Compute the final Bézier control points P_a and P_b: they each correspond to an extension of 4/3 of the line connecting the start/end point and the intersection points computed in step 3.

The process is illustrated on Figure 4.12.

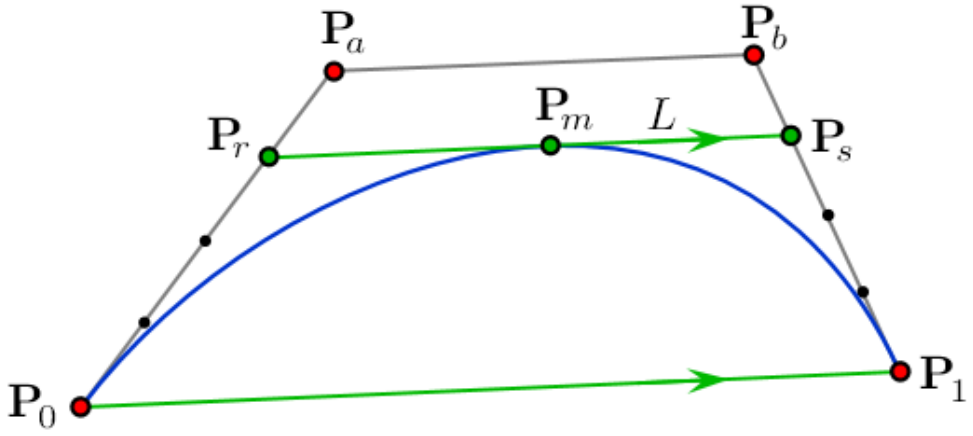

Figure 4.12: Illustration of the computation of the Bézier control points [15]

With this approach, the computed Bézier control points provide a good approximation of the true curve. The difference between the approximation and the true curve is shown on Figures 4.13 and 4.14.

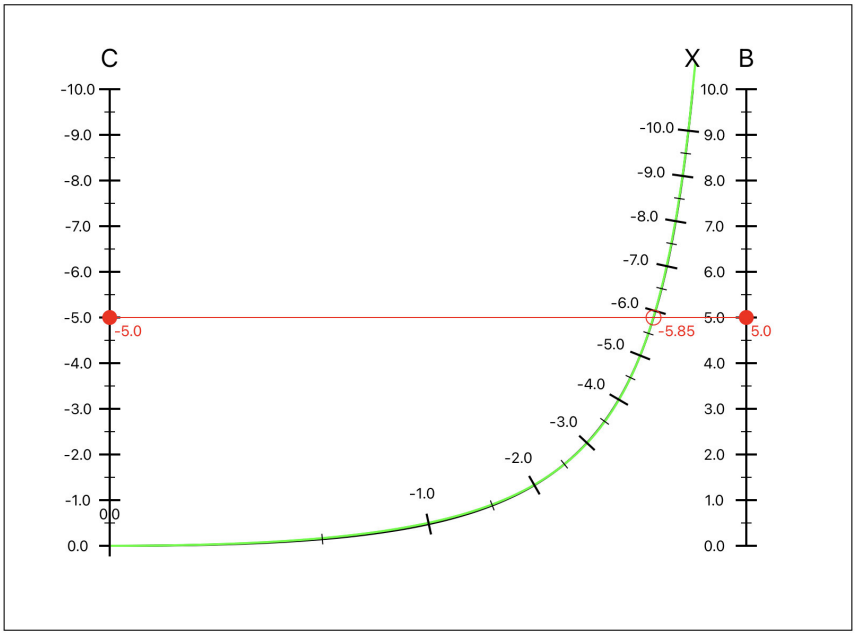

Figure 4.13: Difference between the true curve (green) and the approximated curved (black Bézier curve) at initial zoom level

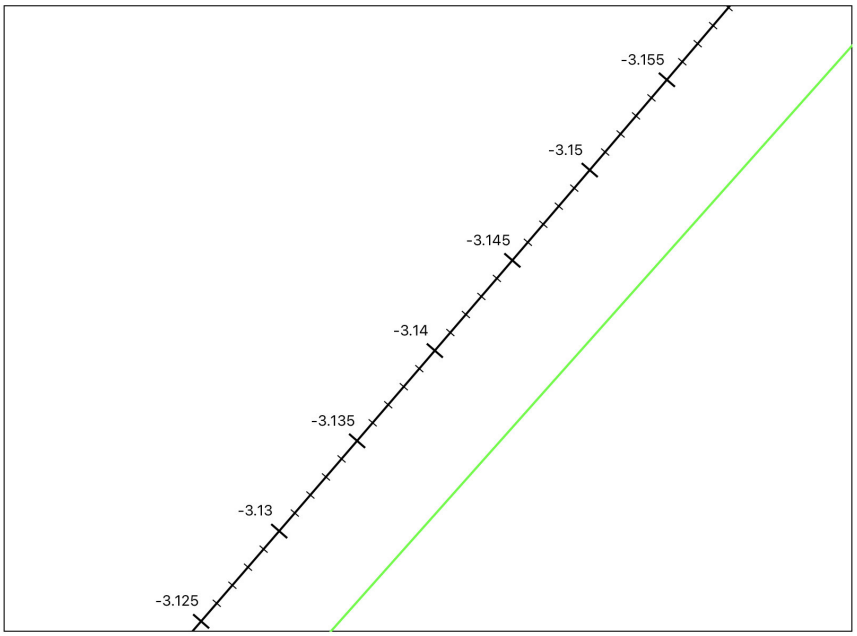

Figure 4.14: Difference between the true curve (green) and the approximated curved (black Bézier curve) with a high zoom level

As we can see, at the initial zoom level the difference is barely noticeable, but it increases at high zoom level, causing problems of precision: the value of the X variable is not exactly at the correct place. This problem could be solved by using multiple Bézier curves together, each one responsible for a portion of the curve.

The **getPoint** function for the curved scale is pretty straightforward: once the scale has been built, invoking the function computes a pair of (B,C) values that gives the requested value for X w.r.t equation (4.8). Then, the position associated with that value is obtained by computing the intersection between the line connecting the pair of values (B,C) and the Bézier curve.

4.4 User interaction

This section explains in detail how the user interactions on the nomograms (panning and zooming) are implemented and how important optimizations have been set up.

4.4.1 Panning

When the user drags his finger accross the topView, an event called Panned Gesture Recognizer is automatically triggered. It records both the location of the finger as well as the panning movement it performs. For the translation of the nomogram, the location of the finger does not matter: the useful information is the translation vector returned by the gesture recognizer. In order to translate the nomogram, this translation vector is applied to the start and end coordinates of each scale, to their Bézier control points in case of a curved line, and to the position of each graduation. This way, all these points have to be computed only once, the translation only modifies their existing coordinates.

4.4.2 Zooming

Similarly to panning, when the users pinches his fingers on the screen, an event called Pinch Gesture Recognizer is automatically triggered. It records the position of the center point between the fingers as well as the scale of the pinching movement. The scale is a floating value, if it is smaller than 1 it indicates a "dezooming" movement, which is when the users brings his fingers together. Inversely, a scale value greater than 1 indicates a "zooming" movement, which is when the users spreads his fingers. Unlike the panning event, both the location and the scale of the zoom event are useful to perform the zoom on the nomogram: for a better user experience, the zoom always occurs at the location of the user's fingers. Similarly to panning, the zoom modifies the coordinates of the scales as well as each of their

graduations. In order to apply this zoom on the coordinates, each point is first translated at the zoom position, then scaled by the scaling factor provided by the gesture recognizer, and then translated back, ensuring that the zoom occurs on the position of the fingers.

4.4.3 Optimizations

Since panning and zooming apply their effects on all the graduations for each scale, keeping the app fluid while the user performs an interaction is very challenging, especially if the number of graduations increase with the zoom level. To prevent the app from being slower and slower the more graduations there are, the list of graduations for each scale is updated dynamically whenever the users performs zooming or panning: each graduation that is not displayed on the screen is removed from the graduation list. This way, at any zoom level, there are always a little number of graduations and the zooming and panning can be performed smoothly. Graduations are automatically re-added to the list if they reappear on the screen, using the current step between graduations to compute which value should be added. These optimizations make a clear difference in terms of performance.

4.5 View syncing

In order to implement the synchronization between the topView and each of the 3 zoomedViews, each scale has a secondary start point and end point (and secondary control points for a Bézier curve), as well as a secondary list of graduations. This allows for an efficient implementation since the scales represented in the topView and each zoomedView belong to the same Scale object. Whenever user interaction is performed on the topView, only the primary start/end points, Bézier control points and graduations are modified. Changing the variable of a value results in a translation on the corresponding zoomedView in order to keep the current variable value at the center of the View. The translation in the zoomedView has the equivalent implementation as the manual translation in the topView that was covered in the previous section, except that it is operated on the secondary start/end points, Bézier control points and graduations. Zooming on any zoomedView is also implemented as in the topView, except that the zoom level is independent for each of the zoomedViews. The same optimizations regarding the graduations are implemented in the zoomed views: any graduation that is not displayed in the View is discarded from the list in order to improve the performance when zooming or panning is applied.

Chapter 5

Future work

This version of NomoPad is a first step towards bringing back nomography in our technological landscape. It has already several interesting features but is far from being a complete app. Here is a non-exhaustive list of features that could be added to future versions of NomoPad:

- **Supports for more types of nomograms**: currently, NomoPad can build nomograms for equations representing any addition or multiplication. It also features a nomogram for a specific equation of the second degree that acts as a "proof-of-concept". There are many more types of nomograms representing a large panel of equations which could be added to NomoPad.
- **Adding nomograms**: currently, all the nomograms featured in NomoPad are hard-coded in the application. The code being very modular, it would be a simple task to add a page where the user could add a new nomogram to the library, based on the supported equation types.
- **Parser to build a nomogram for any type of equation**: with a sufficiently extensive nomogram library, NomoPad could feature a parser that could handle any equation inputted by the user and build the nomogram associated with it by transforming the given equation into the correct nomogram type.

Bibliography

[1] D'Ocagne M. (1899). *Traité de Nomographie*, Gauthier-Villars.

[2] Descartes R. (1637). *La géométrie*, A. Hermann.

[3] Pouchet L. (1795). *Arithmétique linéaire, ou Nouvelle méthode abrégée de calculer, que l'on peut pratiquer sans savoir lire ni écrire*, Rouen : Guedra, an IV.

[4] Allcock H. J., Reginald Jones J. (1950) *The Nomogram - The theory and practical construction of computuation charts*, Sir Isaac Pitman & Sons, LTD.

[5] Otto E. (1963). *Nomography*, translated by Smolska J., Pergamon Press.

[6] Douglass D. R., Adams P. D. (1947). *Elements of Nomography*, Pravana Books.

[7] Epstein I. L. (1958). *Nomography*, Pravana Books.

[8] Apple, *Swift - The powerful programming language that's also easy to learn.*, https://developer.apple.com/swift/, 2023.

[9] Apple, *Programming with Objective-C*, https://developer.apple.com/library/archive/documentatio 2014.

[10] Apple, *XCode*, https://developer.apple.com/xcode/, 2023.

[11] Apple, *Beta Testing made simple with TestFlight*, https://developer.apple.com/testflight/, 2023.

[12] Apple, *Apple Developer Program*, https://developer.apple.com/programs/, 2023.

[13] Wikipedia, *Bézier curve*, https://en.wikipedia.org/wiki/B%C3%A9zier_curve, 2023.

[14] VectorStock, *Ruler cm measurement numbers scale vector image*, https://www.vectorstock.com/royalty-free-vector/ruler-cm-measurement-numbers-scale-vector-22893867, 2023.

[15] Mathematics Stack Exchange, bubba, *Easy way to draw conics with Bezier control points?*, https://math.stackexchange.com/questions/873224/calculate-control-points-of-cubic-bezier-curve-approximating-a-part-of-a-circle, 2018.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**