Matrex

Table of Contents

Matrex is a graphical tool to **show, organize and calculate big amounts of data**.

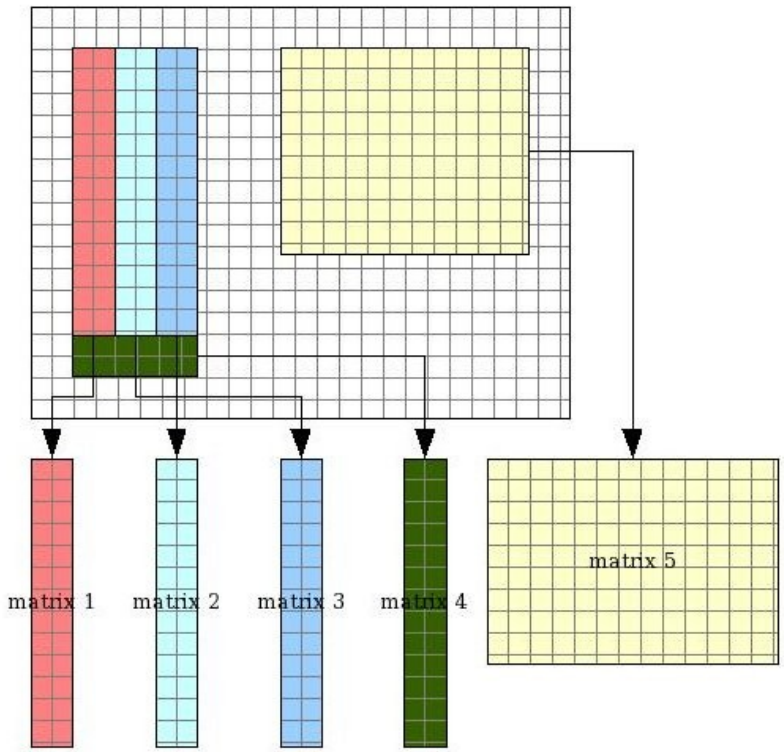
As such, it is in the same area as spreadsheets. Comparing it with spreadsheets, it is specially suitable to complex calculations, for example to test and use mathematical models and for other statistical, engineering, physical, and generally scientific calculations.

The idea that originated Matrex is that a typical spreadsheet is logically formed by:

- vectors (columns or rows of the spreadsheet)
- matrices
- scalars like results of aggregate operations (sums, averages), parameters and constants.

So, instead of reducing everything to cells of a spreadsheet, Matrex handles vectors, matrices and scalars for what they are, under the general concept of Matrex “matrices”: a matrix is an homogeneous rectangular block of numbers, texts, dates or booleans, which can also be composed only by a column or by a single element.

The following example represents a spreadsheet with the parts in different colors (vectors in red, green, blue, matrix in yellow), and how the same content is represented in Matrex:



With Matrex, you edit matrices or read them from external sources (like databases) and use formulas to calculate other matrices.

Clearly, to use Matrex to handle a small calculations involving 10-20 numbers is like to use a cannon to shoot at birds. More complex problems, like statistical, engineering and in general scientific problems are instead well handled.

The idea to represent your problem with matrices can be attractive, but then you need a way to visually summarize the result of your work. For this purpose in Matrex you can show your matrices in something similar to a read-only spreadsheet, called presentation, or in charts.

What is the equivalent of a spreadsheet in Matrex?

A spreadsheet is equivalent to a Matrex **project**, which is a set of items:

- **Matrices** to contain the spreadsheet columns, blocks of numbers and single numbers.
- **Functions** equivalent to the spreadsheet formulas.
- **Presentations** equivalent to the visual part of the spreadsheet.
- **Charts** equivalent to the charts of a spreadsheet.
- **Timers** equivalent to the timers controls in a spreadsheet.

Why I should use Matrex instead of a spreadsheet application?

Modeling your problem as a set of matrices and functions instead of putting all together in a spreadsheet gives more structure to the problem.

That has the following effects:

- **Performance**: formulas are not applied to single cells but to matrices; the system does not need to look at every cell to know the type of the cell, which formula to use, interpret it, look which cells to apply it to.
- **Multi-threading**: the unit is not more the cell or the spreadsheet, but the matrix. Which means that it is possible to use threads when recalculating the matrices of a project, one for each function recalculation. More calculations can be done in the same time and your GUI does not block waiting for the results.
- **Control**: every item (matrices, functions, presentations...) has a meaningful name and is part of a structure that is easy to understand.
  Formulas are often replicated in multiple cells of the spreadsheet, because they apply to single cells. This can easily cause errors. In Matrex functions apply to matrices, so you need to write them only once.
  Spreadsheet formulas are often very complex, which makes them difficult to understand, and hidden in the cells. Matrex gives you the possibility to write complex formulas, but then they are translated in single functions and matrices, with names and (possibly) comments.
- You can easily **reuse** your Matrex project or parts of it (functions) for other purposes. The fact that a Matrex project is composed by small parts (matrices and functions) lets them be reusable. For example you can reuse a complex database query in many projects.
- A Matrex project can be used by an **external application** through the **Matrex API**. In short, you can use Matrex without the GUI as a library (jar) of your application.
  This means that, for example, a spreadsheet expert can write a Matrex project and a developer can build an application that uses that project to make a calculation.
- It will be possible, from version 2.0, to **share** a Matrex project among many users putting it in a **Matrex server**. When a user adds or changes items to the shared project, the other users see these changes in the moment they are made.

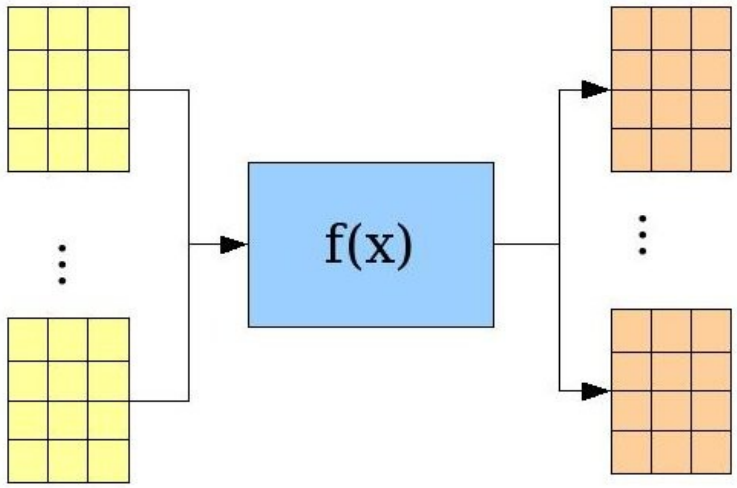
Concepts

- **Matrix**: a rectangular block of numbers, texts, dates or booleans. Can be composed by only one column or row, in which case it is equivalent to a vector. It can be also composed by only one cell, in which case it is equivalent to a single item.
  For performance reasons, Matrex considers vectors the matrices with only one column, not the ones with one row. This means that using matrices with one column gives better performances than using matrices with one row.
  To obtain a vector-like matrix from a matrix with only one row, you need to use the *transpose* function.

  The contained cells are all of the same type (number, text, date or boolean), so you can speak of numeric matrices, text matrices, date matrices or boolean matrices.

  The content of a matrix should be homogeneous, which means that all the contained cells contain the same kind of information, like the values in a column of a database table.

- **Function**: the equivalent of a function used in a spreadsheet formula. Combines matrices to obtain matrices. If one of the input matrices changes, the function is recalculated (exactly how it happens in a spreadsheet).



Functions cannot contain composite formulas.
It is anyway possible to enter composite formulas in Matrex: the **expression parser** translates formulas in a set of functions and intermediate matrices which are added to

the functions and matrices trees in the project**.**

- **Parameters**: some functions need some “non matrix” parameters. For example the “query” function that queries the database and returns the result set as a set of vectors, needs the SQL query text as parameter.
- **Tree**: concept similar to the “files tree” that we are used to see in a file system browser, like for example “Explorer” in Windows and “Nautilus” in Linux. Instead of the files we have matrices, functions, presentations.... The absolute path of the “directory” containing an item is called the **package** of the item, for example “/market/fonds/quotations”.
  In a tree you can add items, delete them, move them from one package to the other, rename them and all the other operations you are used to in a file system browser.
- **Function Template**: When you add a function to a project, you need to build it from a template. The template is in some way the “structure” of the function, without data (the parameters and the input and output matrices).
  A template specifies which kind of parameters, input and output matrices the function needs to work (definitions) and the code that calculates the function.
  For example there is the “plus” function template for the sum of matrices and the “average” one for the average of a matrix.

  You can **add a template** in your machine or in a Matrex server, specifying parameters, input and output matrices definitions, and the code that has to be executed to calculate the function. The code can be written in Java (Matrex is also written in Java) or in a JVM scripting language. For version 1.0 we chosen as scripting language Jython.
- **Thread**: is the unit of execution in which a function transforms its input matrices to the output/result matrices. It can be executed in parallel with other threads, in the same time. Using threads can improve the overall performance of the calculation, because complex functions are calculated in the same time as other functions and don't get the control of the GUI and in general of the whole system.
- **Timer**: executes a function periodically. Used for example to recalculate the database query function every 30 seconds, to capture the changes in the queried tables content.
- **Presentation**: very similar to the concept of spreadsheet, but only used to show the data, not to change it. It is a big grid covered with the matrices and texts you chosen to add to it. When a matrix changes its content, the presentation shows the changes.
- **Chart**: equivalent to a chart in a spreadsheet application, gets some matrices as input and shows them as a 2D (for vector-like matrices) or 3D chart.
- **Project**: set of matrices, functions, presentations, charts and timers used for a common purpose. Matrex works by projects, which means that to work with matrices, functions etc. you need to create a project that contains them.
- **Project item**: a matrix, function, presentation, chart or timer.

- **Machine**: can be a Matrex desktop application (called “local”) or a Matrex server (from version 2.0), used by the desktop applications (clients) to share projects.

System architecture in the future (version 2.0)

By now, Matrex is a desktop application. In the future it will become a system, composed by the desktop application and the server application.

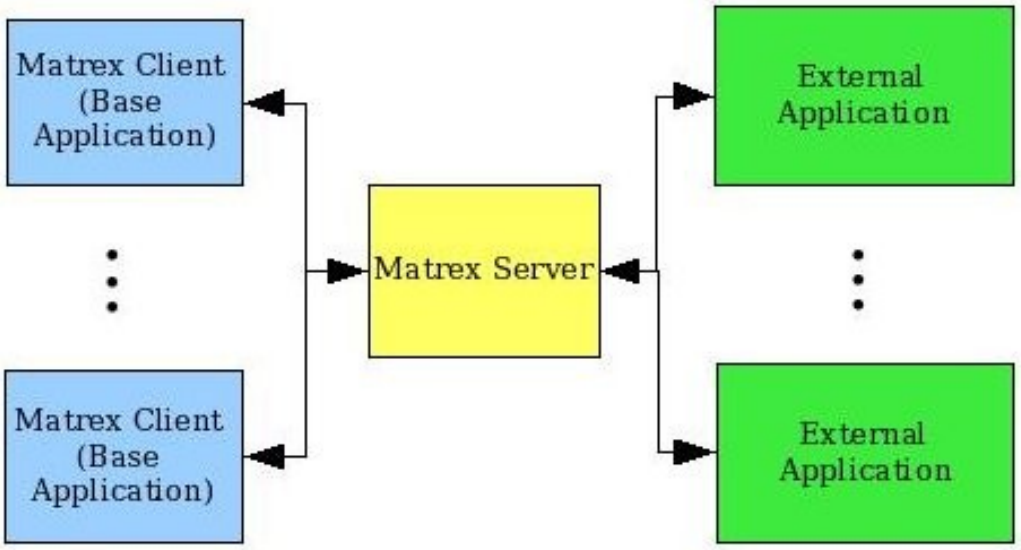
The server application is used to share projects among desktop applications (clients). It is composed by:

- the internal engine that calculates the functions (same as the Desktop application)
- the projects loader and saver (same as the Desktop application)
- a network layer that receives the clients requests and sends back responses and updates.

The Desktop application will be able to work standalone (as it does now) or connected to a Matrex server.

The reasons for a desktop application to connect to a server are:

- **Share Matrex projects** with other users. The project is located in the server and all the clients that access it see when its content changes. For example if clients A and B are connected to the same Matrex server and access the same project, if client A adds a function to the project, the new function appears also in client B. Also, if the content of a matrix changes, it is immediately observable in both the clients.
- Use special **function templates** that are available on the server. New function templates can be added to the system, and it is a good idea to add them to a server, so they can be shared among users.
- **Distribute the computation load**. A project placed in a server is calculated by the server, so the client only needs to interact with the project and let the server do the calculation.
  Also, in the future (version > 2.0) it is possible to think about **distributed projects**: a project in one machine, its subprojects (projects used by the main project to delegate parts of the calculation) in other machines.



Technology

Matrex is written **Java**.

I have considered also the following languages:

- **C++** for the performances.
- **Python** for the fast development.
- **C#** for the GUI.

The reasons because i have chosen Java are:

- C++ development is too slow, building a GUI with it is too complex.
- Python is a script language and I don't seriously think I can use it for a project of this size.
- C# in .Net forces the application to work only in Windows. On the other side I'm not totally sure on the maturity of Mono and its GTK GUI library.
- There are many free reliable APIs for Java, mathematical and to use protocols like RMI, Jini, SOAP, CORBA, JMS.
- I have a long experience programming with Java.

I am aware on the main problem using Java for such a project: the **performances**.
It is sure slower than C++, mainly for the calculations, which are important in this project.

Anyway, these performance differences are not so strong as many think and will be reduced by the progresses of the JIT (just in time compiler), which compiles the code during the execution.

In version 2.0, the servers, without GUI, can be optimized for the best calculation performance.

Moreover, the Desktop application uses **SWT** (Standard Widget Toolkit) as GUI API, which is also used by Eclipse. This API is considered faster than the Java standard GUI library **Swing**, because it uses directly the API of the operating system.