Modelagem do Acoplamento Poço-Reservatório com Variação de Propriedades Termodinâmicas em Reservatórios com Alto Teor de CO_2

Pires, A. P. Ortiz, C. E. P.
Bueno, A. D. Remigio, S. A. E.

Laboratório Engenharia Exploração e Produção de Petróleo
Universidade Estadual do Norte Fluminense Darcy Ribeiro

Software Development Approach

Pedro Henrique Linhares

Universidade Estadual do Norte Fluminense Darcy Ribeiro
Laboratório de Engenharia e Exploração de Petróleo

Macaé, June 2014

Outline

Sumário

Definition

- A programming language is an artificial language designed to communicate instructions to a machine, particularly a computer. Programming languages can be used to create programs to control the behavior of a machine or to express algorithms.
- Over the years, hundreds of programming languages have been created with different sintaxes and purposes. As the time passed, they began being classified into generations.

Programming languages generations

- Today, these languages can be classifed into five programming language generations. Historically, this classification was used to indicate increasing power of programming styles.
- As higher the language gets in the generation's scale, more layers of abstraction are added above the machine-code program.

First Generation

- Use a binary code that consists of strings of only zeroes (0) and ones. (1). The use of binary code is very difficult to learn and use.

First Generation

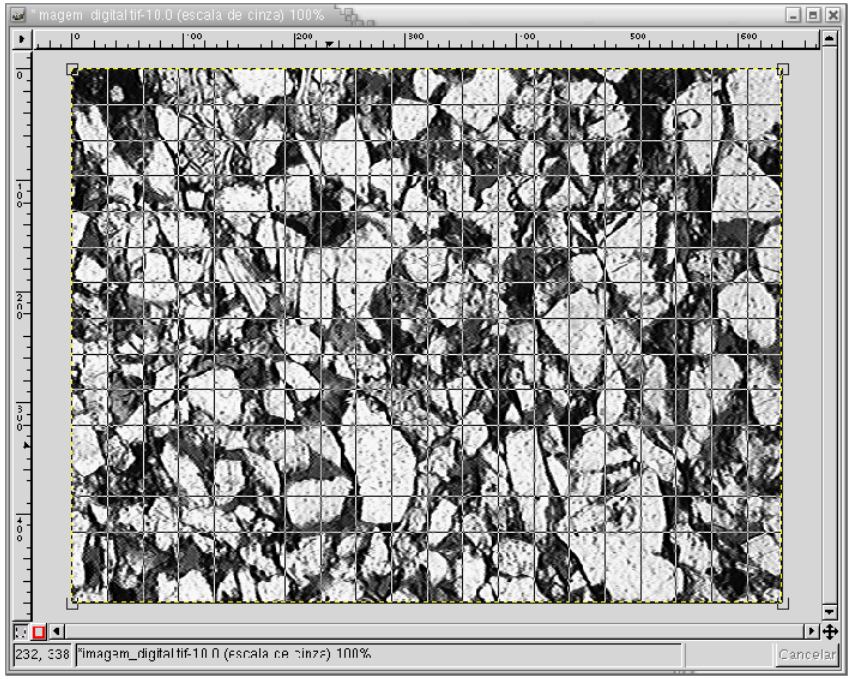


Figure: Example of binary code

Second Generation

- These are Assembly languages that use mnemonics code that consists of very short words for commands.
- Assembly language programs need to be converted into machine language by an assembler before it can be run.

Second Generation

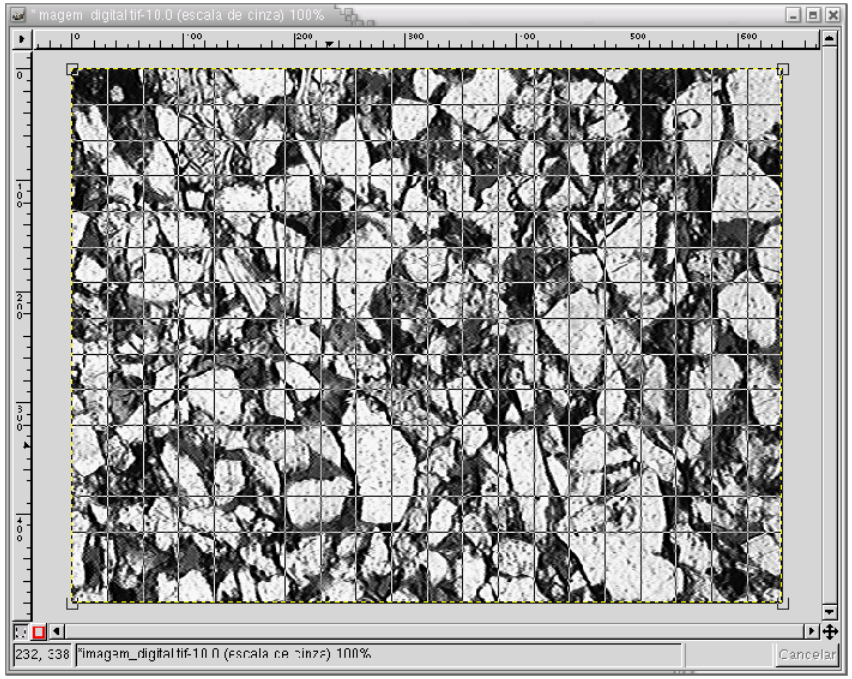


Figure: Hello World in Assembly

Third Generation

Introduced data and control structures . Third generation languages are portable or machine independent where a program written from one computer can also be used by another computer however the source code must be recompiled by a language compiler by the other computer.

Fortran, Algol, Pascal and C are examples of this. As they are compiled languages, the compiled code is a first generation machine code and for being so close to what computers can understand, the executable code can run very fast.

Third Generation

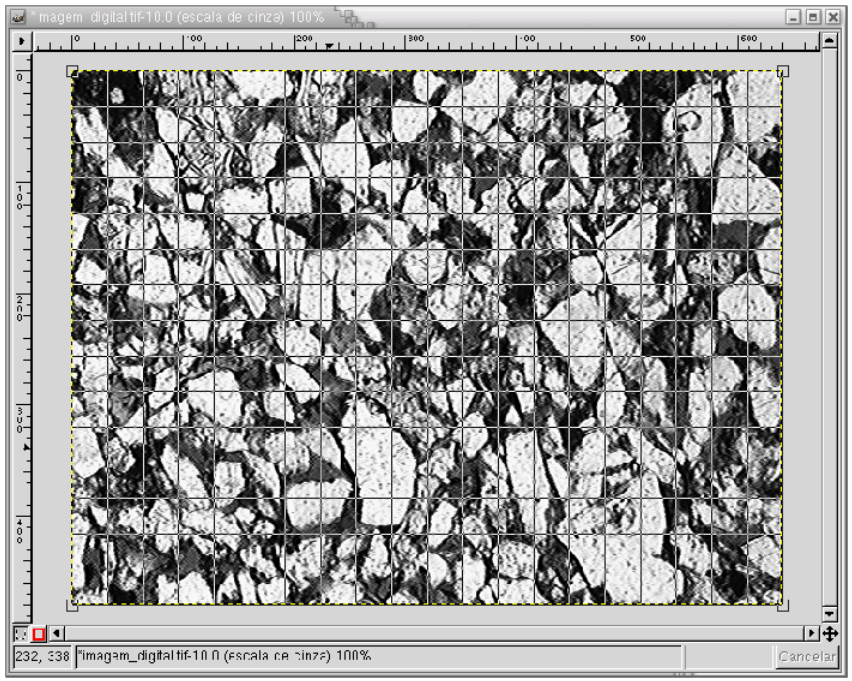


Figure: Hello World in Fortran

Fourth Generation

A fourth generation programming language is a grouping of programming languages that attempt to get closer than 3GLs (Third Generation Language) to human language, form of thinking and conceptualization.

These programming languages are usually interpreted and often used for prototyping. Most of interpreted language implementations execute instructions directly, without previously compiling a program into machine-language instructions. Examples are: Python, Ruby, Perl, Matlab, Tcl, etc.

Fourth Generation

However, by not being previously compiled, they can reduce programming effort, the time it takes to develop software, and the cost of software development.

Given the right problem, the use of an appropriate 4GL can be spectacularly successful and the productivity gains can get as high as 8 times faster than in a 3GL.

Fourth Generation

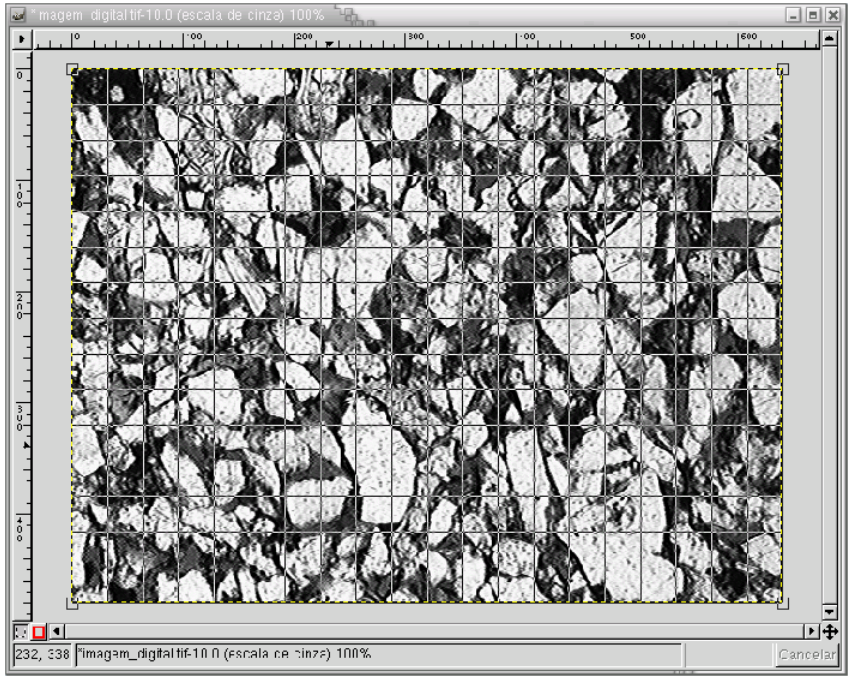


Figure: Hello World in Python

Fourth Generation

As you add more layers above the machine and get closer to the programmer, you write a more concise and straightforward code although losing performance.

Fifth Generation

Is a programming language based on solving problems using constraints given to the program, rather than using an algorithm written by a programmer.

They are designed to make the computer solve a given problem without the programmer.

Includes constraint-based and logic programming languages and some declarative languages. Prolog, OPS5 and Mercury are examples.

Sumário

Basic Approach

Considering those different types of languages, and while trying to get the best results in development and processing time, we have come to the conclusion that there is value in using an intrepreted fourth generation language to prototype the software and after having the final version, implement the code in a compiled language such as Fortran and C++.

Basic Approach

It has proven to be valuable because as we were not so familiar with the subject we are researching, with a interpreted language, we could go straight to start learning and solving the problem in a easier and simpler process.

Basic Approach

Ultimately, in the software development process, our goal is to get from point A to point B, which is our final product, as seen in the Figure below:

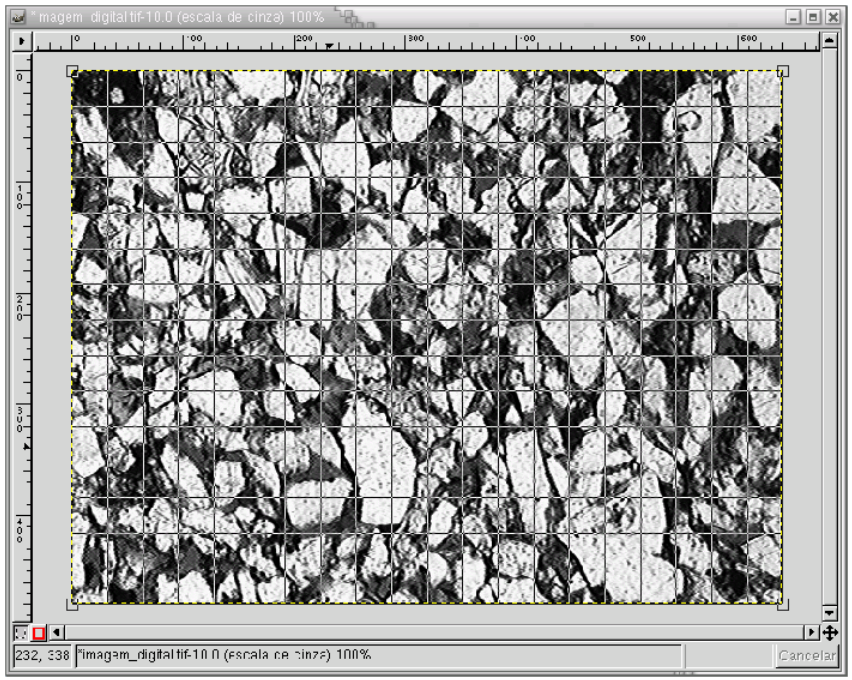


Figure: Software goal

Basic Approach

However, along the way, we end up facing several problems and sometimes being stuck at dead end points as seen below:

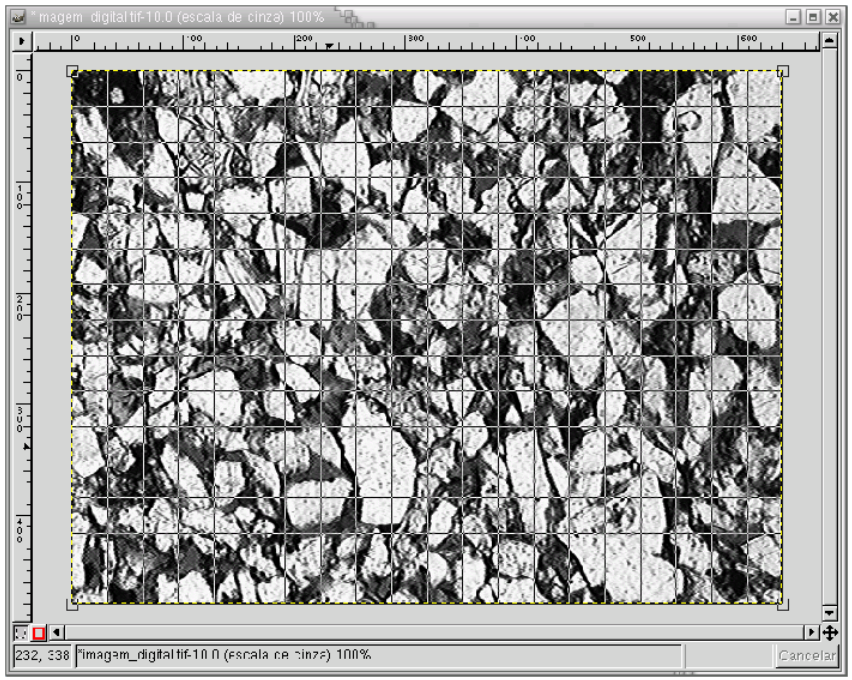


Figure: Dead ends in the software development process

Basic Approach

With prototyping, we can reduce drastically the time spent with this kind of problem.

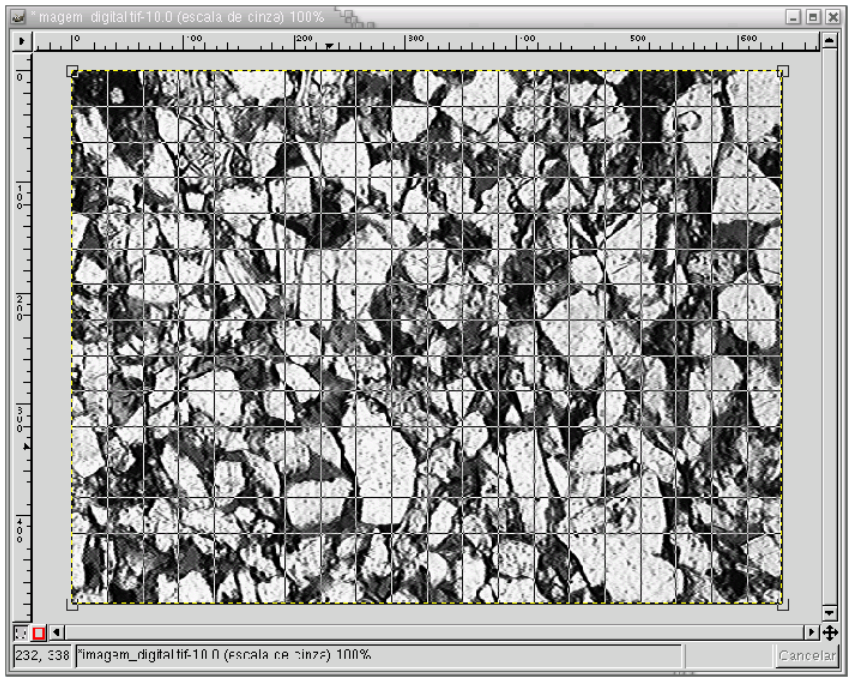
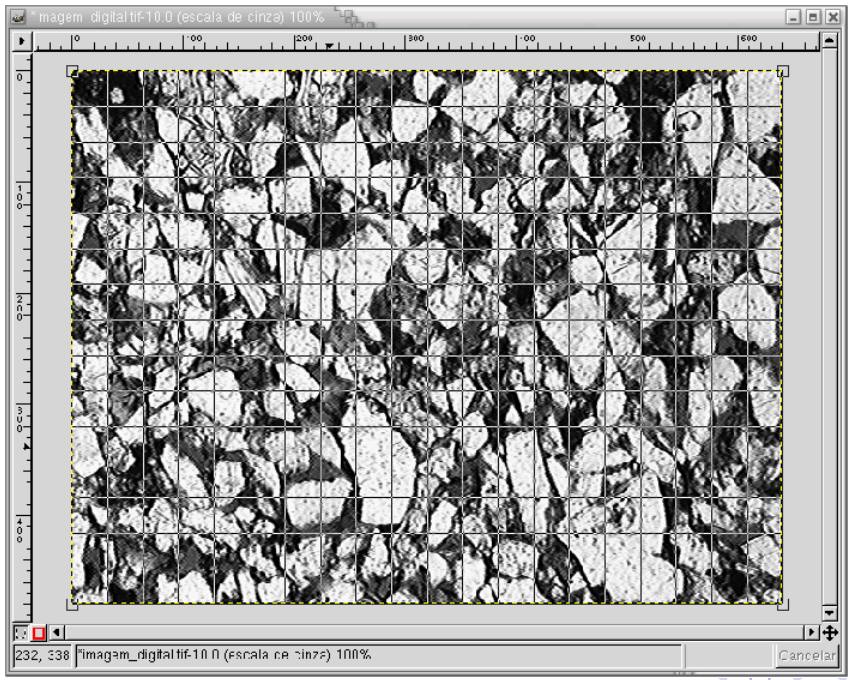


Figure: Areas improved by prototyping

Basic Approach

However, we believe that will come a time, where due to the slower time execution, it will be better to implement directly into a 3GL language like Fortran/C++ as execution time can get as high as days to finish the execution of a test case. But at that point, knowledge about the problem will be much higher.



Python/MatLab and Fortran/C++

With all being said, we have been using Python and MatLab to prototype because it is fast to develop and has a good amount of scientific libraries available. We have also been using Fortran 2003 and C++ to implement code that needs to have high performance in processing time.

Problem

Solve the single-phase, single-component flow of a fluid in a pipe, considering the axial flow direction.

- Single-phase flow;
- Monocomponent fluids;
- There is no heat exchange between the environment and pipe;
- Heat flow.

Fortran

What has been used to develop this code:

- Classes and objects;
- inherintance;
- polymorphism;
- Dynamic allocation;
- Modules;
- UMFPack (set of routines for solving sparse linear systems).

Fortran vs Python

Lines of code:

- Pyhon: 2400 lines
- Fortran: 4000 lines

A Total of 40% more code.

Fortran vs Python Performance

The same example was run in Python and in Fortran. The Fortran code executed on a average of 20x faster.

Testing with a Purpose

Testing of individual software components or modules. Typically done by the programmer, as it requires detailed knowledge of the internal program design and code. The overall objective to not to find every software bug that exists, but to uncover situations that could negatively impact the usability and/or maintainability, while also, helping the programmer identify areas of code that could be responsible for a bug.

Why to test?

Some advantages of software testing:

- Verify that code presents the expected output;
- Verify that we are doing the right software that follows the user requirements;
- Guarantee manutenability and safety while chaging the code as it grows;
- Finding bugs becomes much easier.

Why to test?

In a big software, when a bug occurs, it is usually very hard to track down the source of the problem. The programmer would have to try to isolate parts of the code and verify if it outputs correctly in order to surround the issue.
Automated software testing can help the developer to identify problematic code in several levels depending on the type of test implemented.

Sumário

Types of tests

There are several types of test, the most important are:

- Unit testing: tests the smallest code units (subroutines and functions);
- Integration testing: tests the integration among modules and verify the interfaces between components;
- System testing: tests a completely integrated system to verify that it meets its requirements;
- Acceptance testing: When the system is delivered to the final user for acceptance.

Unit testing

The type of code we are mainly using in this project. We have been using to test the python classes and also the Fortran classes.

Unit testing

It works by doing basic assertions about the code, examples are:

- assert a expression is true or false;
- assert that two values are equal;
- assert a number is equal within a interval;
- assert that two arrays are equal.

The programmer should try to make its tests more comprehensive as possible.

Sumário

Examples

Example of a python class implementation:

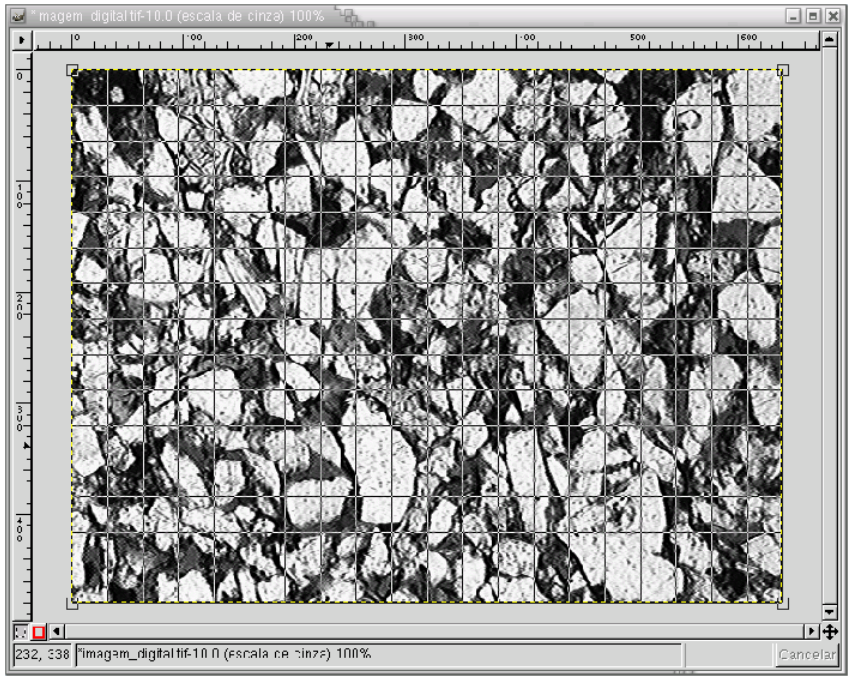


Figure: A Python Class

Examples

The corresponding test case for the Python class:

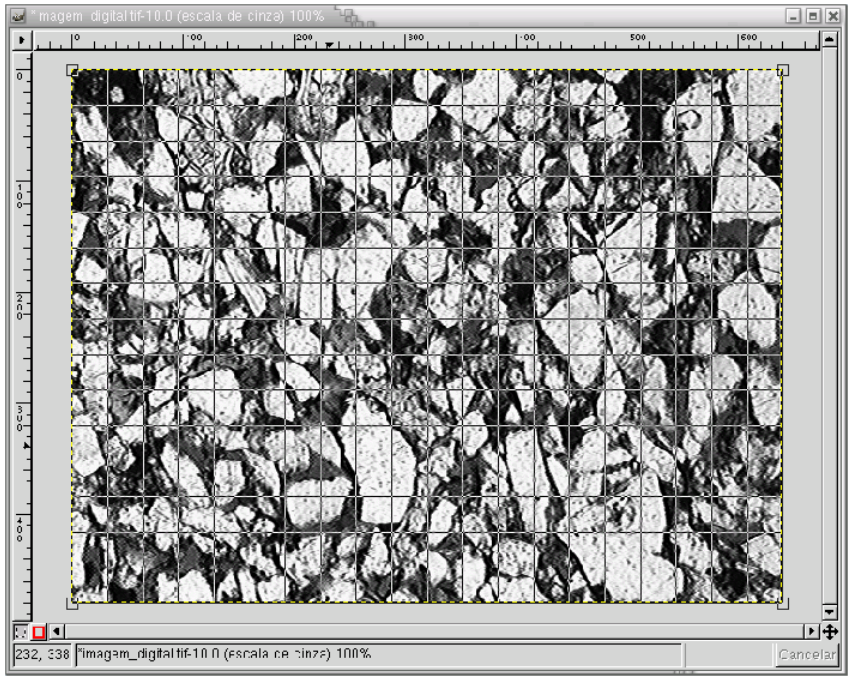


Figure: A Python unit test

Fortran example

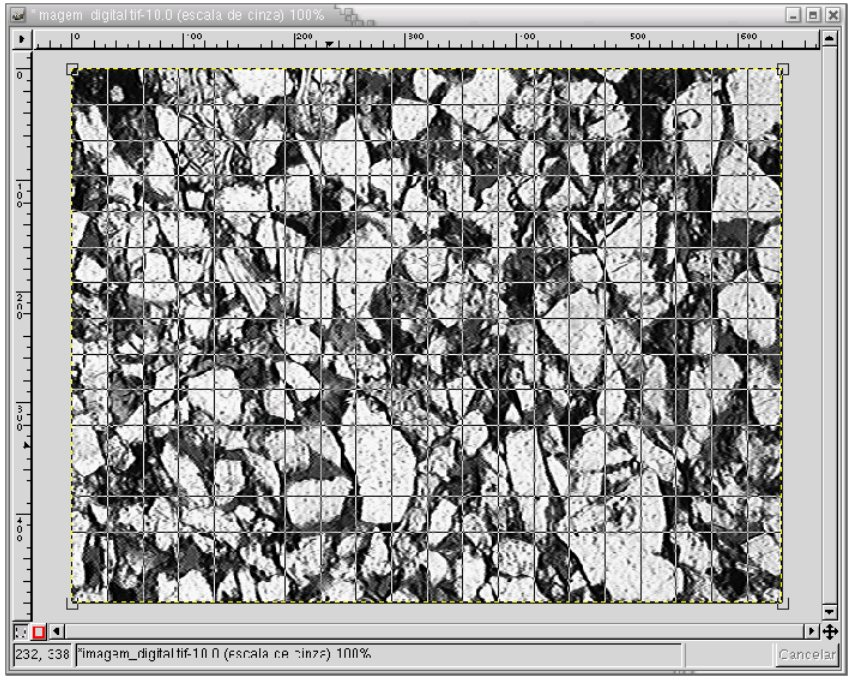


Figure: Fortran 2003 methods

Fortran example

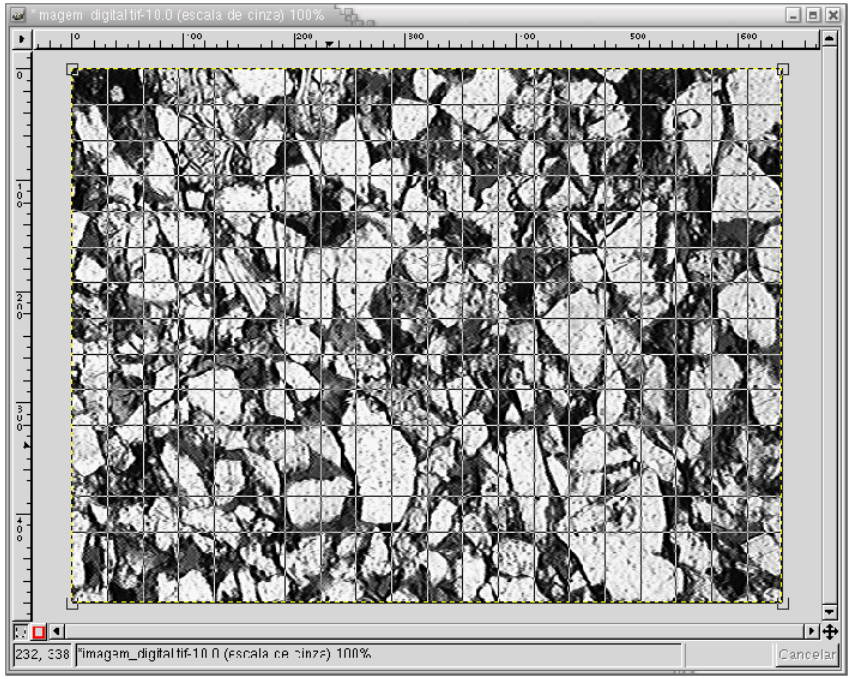


Figure: Fortran test case

Sumário

Thermodynamics

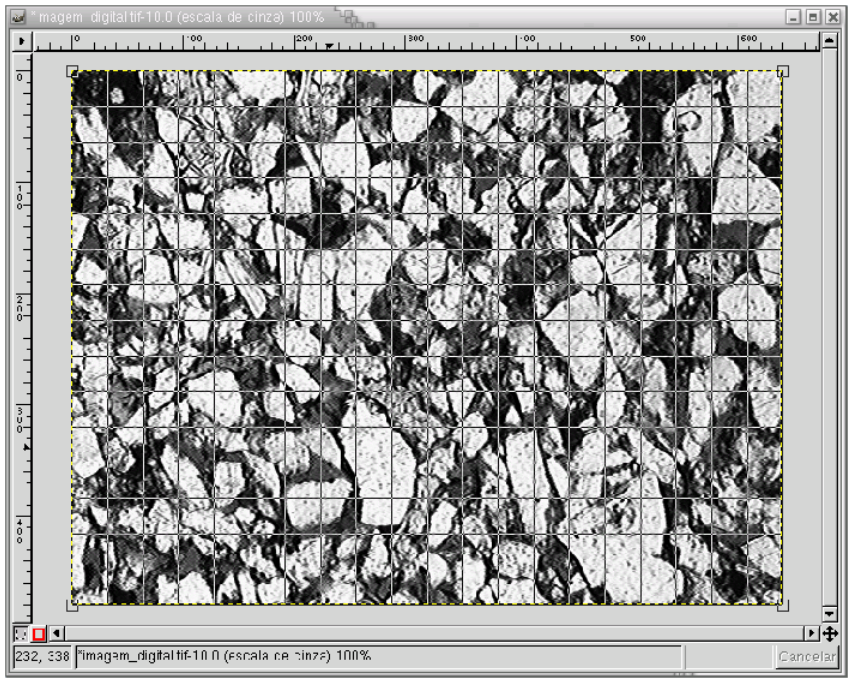


Figure: Thermodynamic's module development overwiew

Sumário

Well - Implementation with Prototyping and Code Conversion

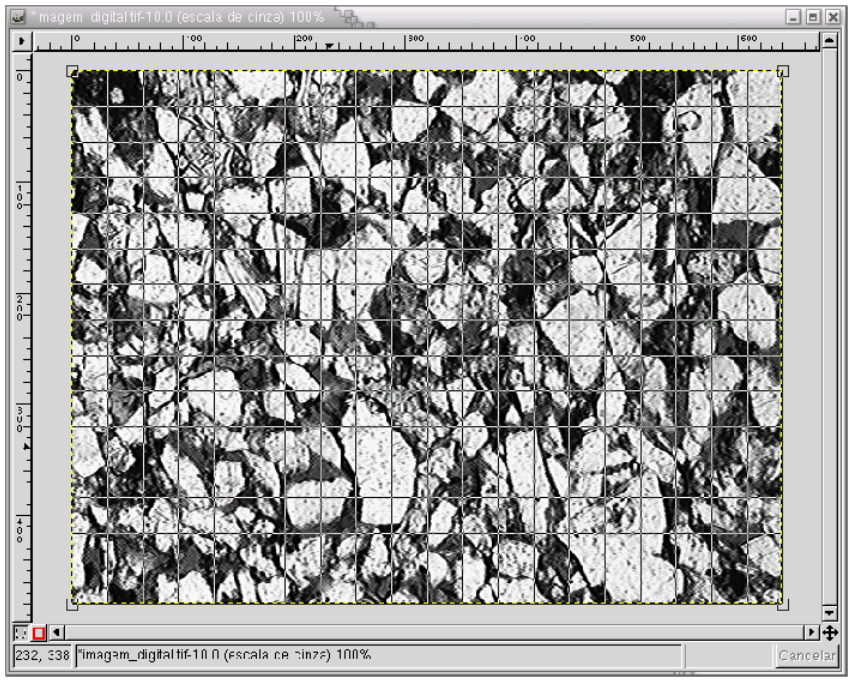


Figure: Well's module development overwiew

Well - Implementation with Prototyping and Code Conversion

Advantages:

- Faster to get to a functional version, generate graphs, get initial results, etc.

Disadvantages:

- Need to rewrite the code;
- executable code is slower.

Sumário

Termodynamics and Well

At the end of development, we expect to have this software packages configuration:

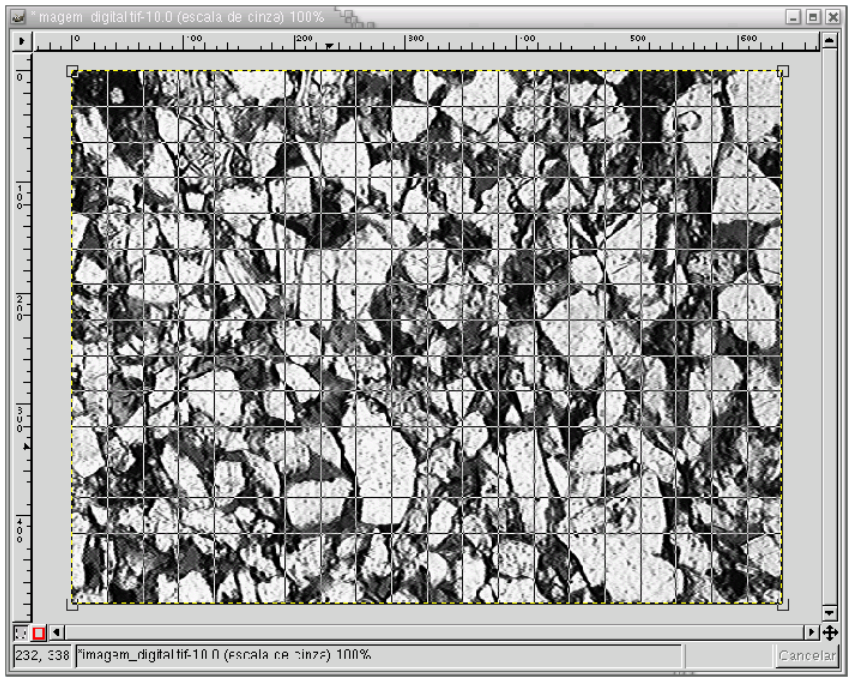


Figure: Final Packages Integration

Future activities

- Finish the numerical modeling;
- Find the best way to integrate all this different languages, and make they communicate;
- Finish implementation:
  - Matlab/Python/Fortran/C++;
- Test integration between the generated libraries and PVT-Phase-GUI.

Acknowledgements