DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

Relatório do Projeto de LAPR1

Análise de Séries Temporais

Equipa
1191507 _ Bárbara Diana
1191430 _ Fábio Fernandes
1191047 _ Rui Ribeiro
1191053 _ Sara Borges

Docente(s)/Orientador(es)
Ana Barata (ABT)
Alexandre Gouveia (AAS)
Ana Moura (AM)

Cliente
Carlos Ferreira (CGF)

Unidade Curricular
Laboratório/Projeto I

Dezembro de 2019.

isep Instituto Superior de Engenharia do Porto

ÍNDICE

1. Introdução

O trabalho descrito neste documento foi-nos proposto no âmbito da cadeira de LAPR1, e tinha o intuito de que os alunos do primeiro ano de Engenharia Informática pudessem pôr em prática os conhecimentos adquiridos nas cadeiras lecionadas ao longo do primeiro semestre.

Posto isto, a tarefa que nos foi dada consistia na elaboração de uma aplicação em linguagem Java que conseguisse analisar séries temporais, bem como estimar os seus valores futuros. Isto é concretizável devido à possibilidade de se analisar uma série temporal através de fórmulas de filtragens (sendo que existem dois tipos de filtração: uma a partir da Média Móvel Simples e outra a partir da Média Exponencialmente Pesada) e de previsões (existindo também dois métodos semelhantes aos da filtragem).

Para executar este projeto, foram-nos disponibilizadas duas semanas focadas no desenvolvimento do trabalho e no esclarecimento de dúvidas através de:

- Oficinas de trabalho, focadas no progresso do projeto em equipa;
- Reuniões com o cliente, onde era possível esclarecer dúvidas com o próprio cliente;
- Aulas Prático-Laboratoriais, nas quais estavam presentes professores especializados em Algoritmia e Programação ou na área das matemáticas que nos iluminaram em relação a fórmulas ou metodologia do trabalho em Java.

Passadas estas semanas, submetemos, no dia 22 de dezembro, o nosso projeto para que fosse avaliado. Posto isto, o que nos restava fazer era elaborar o relatório e preparar a apresentação do trabalho.

2. Metodologia de Trabalho

2.1 EduScrum no desenvolvimento do Projeto

A metodologia *eduScrum*, utilizada durante este projeto, consiste na criação de equipas e atribuição de papéis a cada um dos seus membros. Cada elemento possui um importante papel para a plena realização e concretização dos objetivos, ao qual chamamos como um conjunto de Sprints.

Um dos membros, que não faz propriamente parte da equipa, é o *Product Owner*, que no caso do *eduScrum* é o professor. Este define o que é necessário ser aprendido, controla o funcionamento das equipas criadas e os seus resultados e por fim, avalia os resultados obtidos, de acordo com critérios de avaliação definidos anteriormente. Consoante a avaliação, o *Product Owner* é responsável pelas atribuições de classificações e respetivas aprovações ou reprovações.

Outro dos membros é eleito como *Scrum Master* e este tem a função de coordenar a equipa, planear o que vai ser feito e definir as tarefas, bem como quando devem estar terminadas. Este também desempenha a função de líder, garantido que a equipa atinja o melhor desempenho possível, fornecendo a máxima ajuda possível. Após essa etapa, as tarefas são subdivididas pelos membros dos grupos para que as possam efetuar dentro dos prazos delineados.

No caso do nosso grupo, escolhemos logo o *Scrum Master* que se manteve durante os dois *sprints* realizados (desenvolvimento do trabalho das séries temporais e respetivo relatório), o que garantiu o bom funcionamento do grupo havendo comunicação entre todos os membros e realização das respetivas tarefas, dentro do tempo previamente definido. O máximo empenho de todos permitiu-nos diminuir ou excluir qualquer tipo de erros que poderiam existir, o que mais uma vez levou à otimização do produto final, vindo daí a importância de um bom planeamento, permitindo que tudo ocorra conforme o pretendido.

Esta metodologia que adotámos, foi muito benéfica para a fase de aproveitamento e aprendizagem, pois permitiu a obtenção de ferramentas de trabalho que poderão ser úteis em futuros projetos. Entre essas ferramentas, encontram-se: a capacidade de discussão em grupo; a marcação de datas de entregas; a organização de todos os componentes do projeto; as melhorias na nossa autonomia e responsabilidade; entre outras, que, como se sabe, serão bastante importantes para o nosso futuro, independentemente da área em que nos encontremos.

2.2 Planeamento e distribuição de tarefas

A ferramenta que mais nos foi útil para a organização e distribuição das tarefas foi o *Trello*, onde registámos diariamente a evolução do projeto: as etapas concluídas, os aspetos a melhorar ou a corrigir e quais os membros que mais se focaram em cada tarefa. Com a utilização desta plataforma, foi-nos possível implementar *due dates*, isto é, datas limite para fazer determinada tarefa, de forma a conseguirmos orientar-nos tendo em conta o tempo restante para finalizar o trabalho na sua totalidade.

A distribuição das tarefas foi feita no final de cada dia de trabalho, ao fazer o balanço da evolução do projeto. Posto isto, ou por via *Messenger* ou discutindo pessoalmente, definíamos as tarefas a serem elaboradas, bem como os membros responsáveis por cada uma delas.

2.3 Reflexão crítica sobre a dinâmica do grupo

Desde o início valorizámos a organização. Para que possamos obter resultados o mais positivos possível, os objetivos têm de estar bem definidos dentro do grupo. Por isso, achámos mais eficiente fazer um registo diário de todos os progressos feitos. Este foi um passo fundamental para criar boas práticas de organização no desenrolar do projeto.

Passando à comunicação, podemos afirmar ter sido um grupo em que não faltou boa disposição, empenho e cooperação entre todos os membros. Ao longo do projeto existiram, obviamente, momentos de maiores dificuldades, mas que conseguimos ultrapassar devido à fácil comunicação entre nós, elementos da equipa.

Existem sempre aspetos a serem melhorados tanto a nível individual como enquanto membros de uma equipa composta por pessoas com metodologias de trabalho e formas de pensar diferentes. No entanto, consideramos estes aspetos simplesmente acessórios, não tendo afetado a execução global do projeto.

Resumindo: analisando o progresso do projeto na sua globalidade, achamos que a dinâmica e as relações entre os elementos do grupo foram bastante positivas, o que proporcionou um bom ambiente de trabalho e a conclusão das tarefas atempadamente.

3. Análise de Séries Temporais

Como foi referido anteriormente, o nosso projeto consistia em criar uma aplicação que, com base em dados estatísticos pudesse esboçar uma série temporal e através dela, fazer uma análise da mesma.

A análise iria consistir em utilizar técnicas de suavização e filtragem implementando também, para uma melhor compreensão, a remoção de ruídos e a identificação de possíveis tendências que a série poderia tomar. Também é possível prever valores futuros que as séries poderiam tomar utilizando técnicas de previsão onde, através de um modelo matemático, é praticável capturar o processo que gerou a série temporal, permitindo assim, prever valores futuros da série utilizando dados anteriores.

Nada disto se tornaria possível se não houvesse um estudo e uma pesquisa relacionada com o tema, no intuito de recolher a maior e melhor quantidade de informação para desenvolver o nosso projeto. Desta forma, iremos, de seguida, tratar toda a teoria e proporcionar uma melhor compreensão sobre o tema abordado neste tópico.

3.1. Séries Temporais

Uma série temporal é uma sequência de observações ordenada cronologicamente que, em geral, são recolhidas em intervalos regulares. A análise de séries temporais pode ser aplicada a qualquer variável que varie em função do tempo sendo que, de um modo geral, as observações mais próximas têm também valores mais próximos que se relacionam com os valores mais distantes (Hyndman & Athanasopoulos, 2014; SEMATECH, 2019).

A análise de séries temporais é de grande utilidade em vários domínios como a Música, a Química, a Energia, entre outros, sendo que do seu processamento podem resultar lucros significativos para o conhecimento do negócio ou planeamento de atividades.

Em seguida, apresentam-se exemplos de séries temporais:

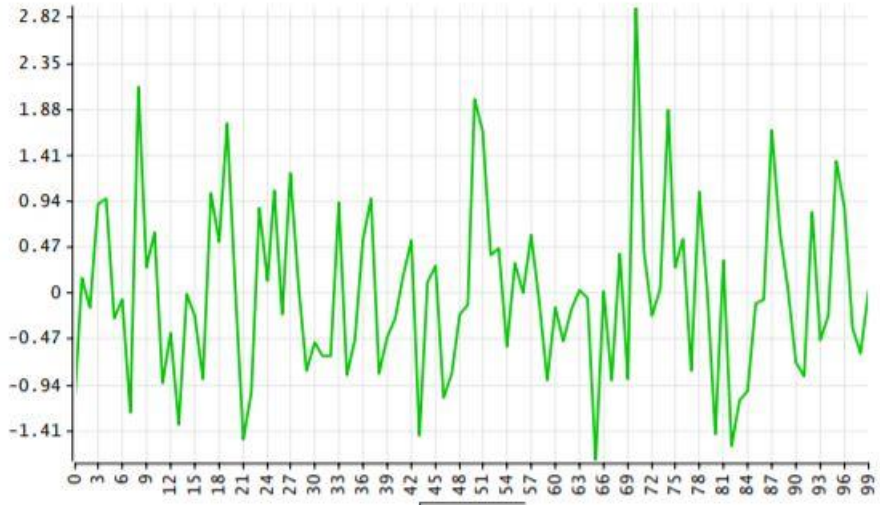

Figura 1 - Ruído Branco Gaussiano

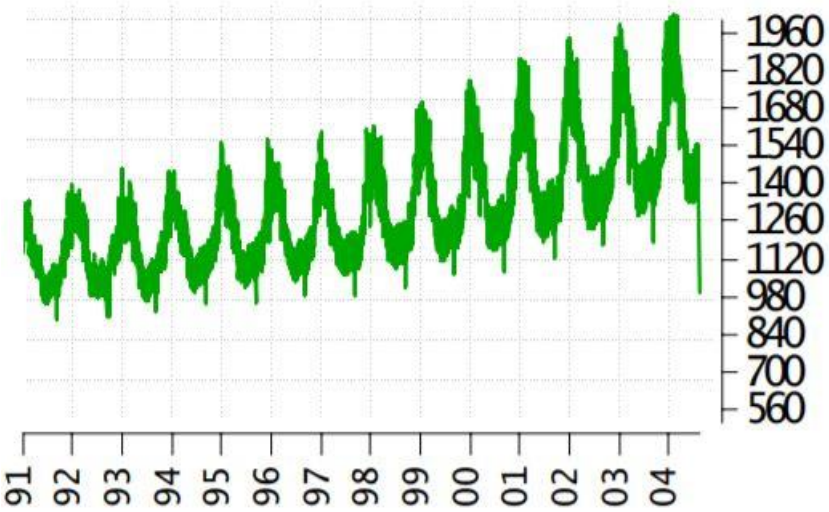

Figura 2 - Consumo Energético

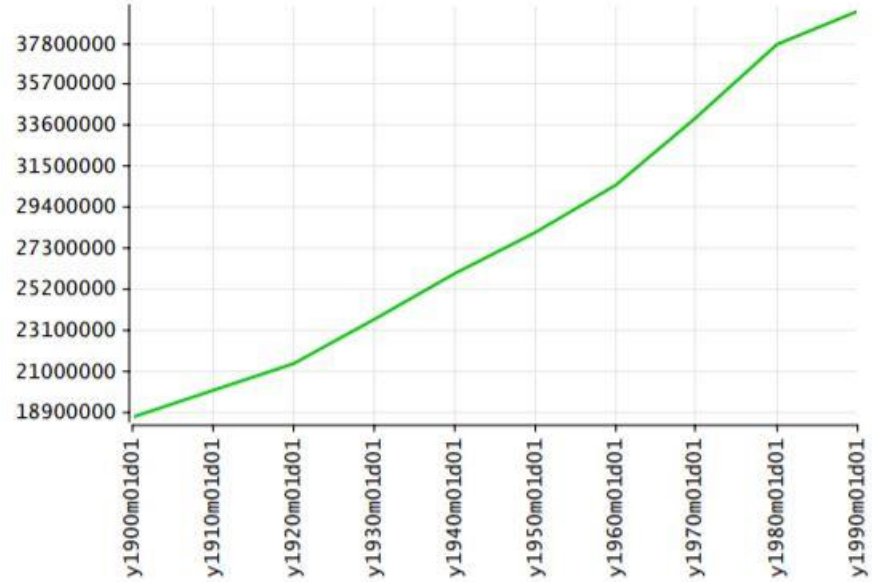

Figura 3 - População de Espanha

Ao analisar uma série temporal, espera-se que nela exista uma causa relacionada com o tempo, que influencia os dados e que possa continuar a influenciá-los no futuro.

3.2. Objetivo das séries temporais

Segundo MORETTIN e TOLOI (1981) e CHATFIELD (2003), os objetivos de analisar uma série temporal são os seguintes:

1. **Descrição**: Determinar as propriedades de uma série como, por exemplo, o padrão de tendência, a existência de alterações estruturais e sazonalidades. Antes de modelar ou prever uma determinada série de tempo, é necessário ter uma ideia preliminar

dos dados, para obter e descrever algumas das suas principais propriedades, o que terá uma enorme contribuição no processo de modelação.

2. **Explicitação ou Modelação**: Encontrar um modelo estatístico adequado que permita explicar o comportamento da série no período observado.
3. **Previsão**: Estimar valores futuros de uma Série Temporal, com base em valores anteriores.
4. **Controlo de processos**: Boas previsões permitem ao estatístico, tomar medidas de forma a controlar um determinado processo.

3.3. Técnicas para análise de séries temporais

Entre as técnicas de filtragem mais utilizadas e simples estão a Média Móvel Simples e a Média Móvel Exponencialmente Pesada (SEMATECH, 2019), ambas utilizadas para efetuar os métodos de suavização implementados na nossa aplicação.

3.3.1. Média Móvel Simples

A Média Móvel Simples (*Simple Moving Average*) é uma média móvel[1] aritmética calculada adicionando valores recentes e, em seguida, dividindo-os pelo número de períodos de tempo na média de cálculo.

As médias de curto prazo respondem rapidamente a mudanças no preço do subjacente, enquanto as médias de longo prazo demoram a reagir.

A Média Móvel Simples é um indicador técnico para determinar se um valor do ativo continuará ou reverterá uma tendência e é definida através da equação:

$$y_i = \frac{1}{n} \sum_{k=0}^{n-1} x_{i-k},$$

onde:

x_i – termos que representam a série original;
y_i – série resultante da aplicação da filtragem;
n – ordem da média móvel.

A Média Móvel Simples é personalizável na medida em que pode ser calculada por um número diferente de períodos de tempo.

[1] Uma média móvel (MA) é um indicador utilizado na análise técnica que ajuda a suavizar a ação do valor filtrando o "ruído" de flutuações aleatórias de valores de curto prazo. É um indicador de tendência atrasado porque é baseado em valores passados.

Quanto maior a ordem da média móvel, mais suave a média móvel simples. Uma média móvel de curto prazo é mais volátil, mas a sua leitura está mais próxima dos dados de origem.

Figura 4 - Exemplo da aplicação da Média Móvel Simples

3.3.2. Média Móvel Exponencialmente Pesada

Uma média móvel exponencial (*Exponencial Moving Average*) é um tipo de média móvel que coloca maior peso e significância nos pontos de dados mais recentes. A média móvel exponencial também é referida como média móvel exponencialmente ponderada.

A média móvel exponencialmente ponderada reage mais significativamente a mudanças recentes de valores do que uma média móvel simples (*Simple Moving Average*), que aplica um peso igual a todas as observações no período.

Como todas as médias móveis, esse indicador técnico é usado para produzir sinais de compra e venda com base em cruzamentos e divergências em relação à média histórica e é definida através da equação:

$$y_i = \alpha x_i + (1 - \alpha) y_{i-1},$$

onde:

x_i – termos que representam a série original;
y_i – série resultante da aplicação da filtragem;
α – constante que varia entre]0,1].

Todas as médias móveis usualmente utilizadas na análise técnica são, por sua própria natureza, indicadores de atraso[2]. Consequentemente, as conclusões tiradas da aplicação de uma média móvel a um gráfico de mercado específico devem ser para confirmar uma movimentação do mercado ou para indicar a sua força.

[2] Um indicador de atraso é qualquer variável mensurável ou observável que se move ou muda de direção depois de uma mudança que ocorre em uma variável alvo.

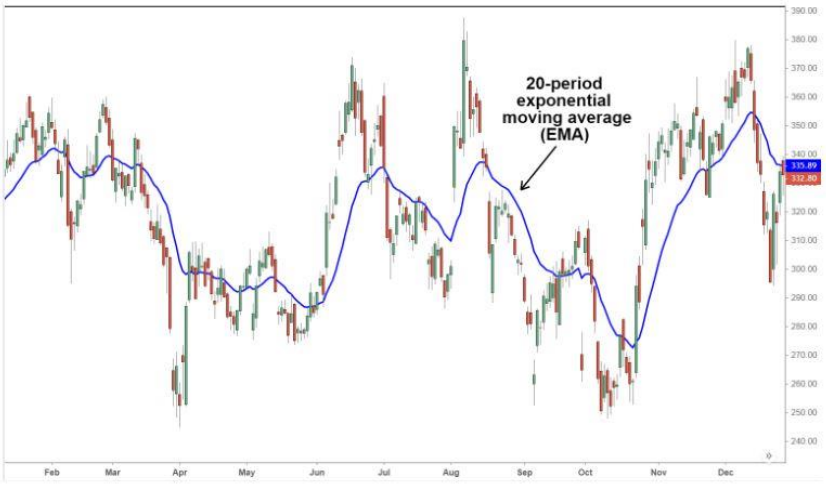

Figura 5 - Exemplo da aplicação da Média Móvel Exponencialmente Pesada

3.4. Previsão

Previsão é um método usado extensivamente em análise de séries temporais para predizer uma variável de resposta como lucro mensal, desempenho dos stoques ou índices de desemprego para um período de tempo especificado. Previsões são baseadas em padrões nos dados existentes.

De forma a responder uma das tarefas propostas do enunciado que nos foi colocado, utilizamos dois métodos de previsão que foram mencionados anteriormente, contudo, com algumas alterações nas fórmulas:

3.4.1. Média Móvel Simples

$$y_{i+1} = \frac{1}{n}\sum_{k=0}^{n-1} x_{i-k},$$

onde:

x_i – termos que representam a série original;
y_{i+1} – representa uma previsão utilizando dados anteriores;
n – ordem da média móvel.

3.4.2. Média Móvel Exponencialmente Pesada

$$y_{i+1} = \alpha x_i + (1 - \alpha) y_i,$$

onde:

x_i – termos que representam a série original;
y_{i+1} - representa uma previsão utilizando dados anteriores;
α - constante que toma valores no intervalo]0, 1].

4. Desenvolvimento e Implementação da Aplicação

Para o desenvolvimento da aplicação, empregámos o conceito de modularização no nosso código, de modo a otimizá-lo e a facilitar a sua leitura e interpretação.

A forma como a aplicação está agrupada bem como a ordem de execução de cada método (através da representação por setas) pode ser evidenciada pelos diagramas presentes no Anexo B.

A organização do código foi feita em torno de 4 requisitos fundamentais: a definição da resolução temporal, a ordenação, a filtragem e previsão e, por último, o esboço de gráficos.

4.1. Definição da Resolução Temporal

O método de definição da resolução temporal baseou-se em definir à *priori* os intervalos de tempo relevantes (24 horas, número de dias num dado mês ou número de dias num dado ano para os períodos diários, mensais ou anuais, respetivamente).

Para cada intervalo de tempo, é efetuada a soma de todos os valores dentro do intervalo, ficando essa soma alocada à primeira posição do array dentro desse período, ou seja, a posição correspondente à primeira hora de cada manhã, tarde, noite, madrugada, dia, mês ou ano, conforme relevante. Este processo é realizado através dos métodos dayPeriod, dailyPeriod, monthlyPeriod e annualPeriod.

```
//calcula consumos diários
public static int dailyPeriod(int[] consumptionMW, int size, LocalDateTime[] dateTime) {
    //NUM_HOURS = nº de horas num dia = 24
    int startPeriod = 0, endPeriod = NUM_HOURS;
    boolean leftovers = false;
    while (endPeriod < size) {
        for (int i = startPeriod + 1; i < endPeriod; i++) {
            consumptionMW[startPeriod] += consumptionMW[i];
        }
        startPeriod = endPeriod;
        endPeriod = endPeriod + NUM_HOURS;
    }
    int leftoverHours = endPeriod - size;
    if (leftoverHours < NUM_HOURS) {
        for (int i = startPeriod + 1; i < size; i++) {
            consumptionMW[startPeriod] += consumptionMW[i];
            leftovers = true;
        }
    }
    size = exchangeInfoDays(consumptionMW, size, NUM_HOURS, dateTime);
    return size;
}
```

Figura 6 – Algoritmo para somar consumos da resolução temporal diária

De seguida, os valores das somas substituem as primeiras posições do array, de forma a ficarem numa disposição sequencial e simplificarem processos posteriores. Tal é executado através dos métodos *exchangeInfoDayPeriods*, *exchangeInfoDays* e *exchangeInfoMonthsYears*.

4.2. Ordenação

No que diz respeito ao 2º pilar – a ordenação – foram colocadas várias opções para a implementação do método de ordenação dos dados recolhidos, sendo eles *Insertion Sort*, *Bubble Sort* e *Merge Sort*.

O método escolhido teria de ser o mais eficaz, ou seja, o que demoraria menos tempo a executar e a organizar a informação que posteriormente tinha sido escolhida. No entanto, para saber qual o mais rápido, necessitaríamos de um teste para o comprovar.

```
package com.mkyong.time;

import java.util.concurrent.TimeUnit;

public class ExecutionTime {

    public static void main(String[] args) throws InterruptedException {

        long lStartTime = System.currentTimeMillis();

        calculation();

        long lEndTime = System.currentTimeMillis();

        long output = lEndTime - lStartTime;

        System.out.println("Elapsed time in milliseconds: " + output);

    }

    private static void calculation() throws InterruptedException {

        //Sleep 2 seconds
        TimeUnit.SECONDS.sleep(2);

    }
}
```

Figura 7 - Método para cronometrar o tempo de execução

Atendendo ao método que nos foi disponibilizado pelo professor de APROG, obtivemos diferentes tempos de execução para cada método de ordenação, sendo eles:

```
bubble - 2444 milisegundos
Insertion - 2104 milisegundos
Merge - 2006 milisegundos
```

Figura 8 - Tempo de execução dos diferentes métodos de ordenação

Como podemos constatar, o *Merge Sort* é sem dúvida o mais rápido a organizar grandes quantidades de informação, que é o caso do nosso projeto.

4.3. Filtragem e Previsão

Paralelamente, desenvolveu-se a metodologia de filtragem e previsão de consumos energéticos, as quais assentaram no desenvolvimento dos módulos MediaMovelSimples e MediaMovelPesada, capazes de ser reutilizados para qualquer uma das metodologias e cuja

função era efetuar os cálculos subjacentes às fórmulas indicadas anteriormente neste relatório e armazenar os valores resultantes em novos *arrays*, como se pode ver abaixo.

```
for (i = n - 1; i < size; i++) {
    for (int k = 0; k < n; k++) {
        mediaMovelSimples[i] += consumptionMW[i - k];
    }
    mediaMovelSimples[i] /= n;
}
```

Figura 9 – Método de cálculo da média movel simples

```
if (isPrevision) {
    mediaMovelPesada = new double[size + 1];
    for (int i = 0; i < size; i++) {
        mediaMovelPesada[0] = consumptionMW[0];
        mediaMovelPesada[i + 1] = (alpha * consumptionMW[i]) + ((1 - alpha) * mediaMovelPesada[i]);
    }
} else {
    mediaMovelPesada = new double[size];
    for (int k = 1; k < size; k++) {
        mediaMovelPesada[0] = consumptionMW[0];
        mediaMovelPesada[k] = (alpha * consumptionMW[k]) + ((1 - alpha) * mediaMovelPesada[k - 1]);
    }
}
```

Figura 10 – Método de cálculo da média móvel pesada

No caso particular dos métodos de previsão, exigiu-se ainda um esforço adicional no sentido de validar datas introduzidas pelo utilizador, certificando que as ditas datas existiam e eram aceitáveis tendo em conta o intervalo no qual se insere a série temporal em análise.

4.4. Gravação dos gráficos em PNG

A nova tarefa consistia na implementação de um método que conseguisse gravar os esboços dos gráficos das séries temporais escolhidas pelo utilizador em ficheiro PNG. A tarefa de encontrar o modo de implementação do método de gravação do gráfico tornou-se numa das nossas maiores dificuldades, até porque, nunca tínhamos tido contacto com a ferramenta sugerida nem com as suas funcionalidades.

A pesquisa foi essencial para o sucesso desta tarefa e na primeira tentativa de solucionar o nosso problema, deparamo-nos com um código que o conseguia solucionar, mas apenas em sistemas operativos como IOS e Linux, sendo que o Windows não permitia executar o comando.

```
ImageTerminal png = new ImageTerminal();
File file = new File("/home/testuser/plot.png");
try
{
        file.createNewFile();
        png.processOutput(new FileInputStream(file));
}
catch (FileNotFoundException ex)
{
        System.err.print(ex);
}
catch (IOException ex)
{
        System.err.print(ex);
}

JavaPlot p = new JavaPlot();
p.setTerminal(png);

p.getAxis("x").setLabel("yield");
p.getAxis("y").setLabel("biomass");
p.getAxis("x").setBoundaries(0.0, 1.0);
p.getAxis("y").setBoundaries(0.0, 1.0);
p.addPlot(setDeleted);
p.addPlot(setExist);
p.setTitle("remaining EMs");
p.plot();

try
{
        ImageIO.write(png.getImage(), "png", file);
}
catch (IOException ex)
{
        System.err.print(ex);
}
```

Figura 11 - Método inicial de gravação de gráficos em PNG

Como o primeiro método se vinha a tornar num fracasso, decidimos colocar questões ao professor de APROG e continuar a nossa pesquisa até encontrar a devida solução.

```
public class HOPE
{

public static void main(String[] args)
{
        String className="parabola";
        String title="teste";
//Generates a file in .png
        File file = new File("IMG/statistics_"+ title + "_" + className + ".png");
//Creates a plot
        JavaPlot plot = new JavaPlot();
//Creates a terminal class that interacts with Gnuplot without showing the graph
        GNUPlotTerminal terminal = new FileTerminal("png","IMG/statistics_" + title + "_" + className + ".png");
        plot.setTerminal(terminal);
//Configurations of axis labels
        plot.set("xlabel", "\"LEI-ISEP\"");
        plot.set("ylabel", "\"" + title + "\"");
        plot.set("autoscale", "ymax");
//Generate the parabola data
        int[][] data = new int[10][2];
        for (int i = 0; i < 10; i++)
        {
                data[i][0] = i;
                data[i][1] = i*i; //Could hava used Math.pow(i,2)
        }
//Appends the parabola to the graphic
        plot.addPlot(data);
//Defines the style of the graph
        PlotStyle stl = ((AbstractPlot) plot.getPlots().get(0)).getPlotStyle();
        stl.setStyle(Style.LINESPOINTS);
//Hides the label on the line
        plot.setKey(JavaPlot.Key.OFF);
//Executes the plot (without showing) and saves the image in IMG directory in your project directory
        plot.plot();
}
}
```

Figura 12 - Método de gravação de uma parábola em PNG

Com este novo código e com as devidas alterações para responder às nossas necessidades, encontrámos a solução para implementar a opção de gravar todos os gráficos que necessitariam de ser esboçados pela necessidade do utilizador, em PNG. Tendo obtido:

```
String title = "Consumo de energia";
//Cria um file em .png
File file = new File("Grafico " + title + ".png");
//Cria um novo plot
JavaPlot plot = new JavaPlot();
//Cria uma classe no terminal que interage com o Gnuplot sem mostrar o gráfico
GNUPlotTerminal terminal = new FileTerminal("png", "Grafico " + "(" + cabecalho + ") " + tipo + " " + agregacao + " " + title + ".png");
plot.setTerminal(terminal);
//Configuração das labels
plot.set("xlabel", "\"Observações\"");
plot.set("ylabel", "\"" + title + "\"");
plot.addPlot(s);
//Define o estilo do gráfico
PlotStyle stl = ((AbstractPlot) plot.getPlots().get(0)).getPlotStyle();
stl.setStyle(Style.LINES);
plot.setKey(JavaPlot.Key.OFF);
plot.plot();

System.out.println("Ficheiro guardado em PNG.");
```

Figura 13 - Método final de gravação de gráficos em PNG

5. Resultados

A versão finalizada da aplicação em Java resultou num programa completamente executável a partir da linha de comandos e disponível em dois modos: interativo e não interativo.

Em ambos os modos, os *outputs* resultantes da implementação da aplicação constituem vários tipos de mensagens na consola, gráficos (com recurso à ferramenta *Gnuplot*) e ainda ficheiros de extensão *png* ou *csv*, com todos os dados desses mesmos gráficos. No modo não interativo, é adicionalmente gerado um ficheiro de extensão *txt* com todo o texto que seria mostrado na consola em modo interativo.

Estando a aplicação dividida em 7 opções de menu, apresentamos abaixo o resultado expectável decorrente da escolha de cada uma:

1. Visualizar gráfico de consumos

Opção que gera uma janela de visualização de um gráfico através do *Gnuplot*.

Em modo interativo, é dada a opção de gravar o mesmo gráfico em formato *png* ou *csv* (podendo o utilizador selecionar um dos formatos, ambos ou nenhum). De acordo com a escolha são gerados os ficheiros adequados.

Em modo não interativo são automaticamente criados ambos os ficheiros.

2. Visualizar média global e distribuição de observações

Opção que gera uma janela de visualização de um gráfico de barras através do *Gnuplot*.

As opções de gravar o gráfico são idênticas às explicadas na opção 1.

Após visualizar e gravar o gráfico, é ainda apresentada ao utilizador a seguinte informação (na consola em ambos os modos e no ficheiro "*Output.txt"* em modo não interativo):

- **Média** – valor da média simples do consumo no período selecionado. Por exemplo, se a periodicidade escolhida para a análise for "mensal" é apresentado o consumo médio por mês.
- **Quantidade de valores próximos da média** – número de observações cujo valor é igual à média ou até 20% superior ou inferior à mesma.
- **Quantidade de valores acima (ou abaixo) da média** – número de observações cujo valor é superior (ou inferior) à média em mais do que 20%.

3. Efetuar uma filtragem

Opção que gera uma janela de visualização de dois gráficos sobrepostos – o original e o resultante da filtragem selecionada.

Após visualizar e gravar o gráfico, é apresentado na consola (e no ficheiro "*Output.txt"* em modo não interativo) o valor do erro absoluto da filtragem em questão.

4. Ordenar valores

Opção que ordena os valores dos consumos periódicos (diários, mensais, etc.) de forma crescente ou decrescente, apresentando de seguida o gráfico dos valores ordenados e, tal como nas opções anteriores, gerando ficheiros *png* e/ou *csv*.

5. Efetuar uma previsão

Opção que calcula o consumo previsto para uma data definida pelo utilizador com base na média móvel simples ou pesada do período imediatamente anterior. O resultado é apresentado sob a forma de texto na consola e, no modo não interativo, em ficheiro *txt*.

6. Alterar o ficheiro

Opção que permite submeter um novo ficheiro *csv* para análise. Se o nome do ficheiro for válido é apresentado na consola o texto "Ficheiro alterado".

0. Sair

Esta opção não apresenta qualquer output e encerra o programa.

5.1 Análise dos resultados

Os resultados efetivos da aplicação foram, em grande parte, de encontro ao esperado.

O formato dos dados de saída foi respeitado no que toca a:

- Gráficos guardados, quer em *png* como *csv*, sendo-lhes atribuído um nome de acordo com o padrão "Consumos (*NomeDaSerieTemporal*.csv) *Opcao TipoDeAgregacao DataDeCriacao HoraDeCriacao*".
- Texto na consola, onde todos os valores são claramente apresentados e seguidos pela unidade **MW** (*megawatts*), quando aplicável.
- Ficheiro de *output* em modo não interativo, constando nele todo o *log* do texto que é apresentado na consola ao longo do correr do programa, sem qualquer modificação.

Os principais desvios do pretendido prenderam-se com o formato dos gráficos, que falharam em apresentar a data das observações no eixo horizontal (ou das abcissas), e a transcrição de informação de gráficos para ficheiros *csv*, onde apenas foi possível incluir a sequência de valores de consumo, isto é, a ordenada de cada ponto do gráfico, não acompanhados pelo respetivo momento da observação (a abcissa).

No que toca ao cálculo e apresentação de valores (como se verifica necessário na opção 3 e 5, por exemplo), apesar do formato do *output* corresponder ao esperado tornou-se difícil aferir a precisão dos cálculos subjacentes uma vez que o ficheiro exemplo no qual se baseou o desenvolvimento da aplicação apresentava várias falhas. Para contornar este obstáculo, na elaboração de testes unitários, tomou-se o cuidado de utilizar apenas períodos sem falhas (que não devem ser confundidos com períodos incompletos, os quais foram testados com sucesso). Ainda assim, admitimos existir a possibilidade de ligeiros erros de cálculo e, consequentemente, do resultado em algumas opções.

No tema de testes unitários, destacamos que foram elaborados 29 testes unitários[3] para 19 dos métodos que estruturam a aplicação. Escolheu-se não elaborar testes para os métodos

[3] Os testes que consideramos mais relevantes podem ser consultados no Anexo A.

não relevantes, ou seja, os de tipo *void* - exceto aqueles cuja função era alterar *arrays* - e os métodos que recorriam à classe *Scanner* para obter *input* de utilizador. Neste conjunto de métodos não testados foram incluídos, em geral, os métodos de transcrição de texto ou gráficos para ficheiro *csv* ou *png*, os métodos de criação de gráficos e ambos os métodos específicos do modo não interativo.

6. Conclusão

Tal como pretendido, conseguimos criar uma aplicação em linguagem Java que analisasse séries temporais: recolhendo os dados necessários de um ficheiro de texto, é capaz de efetuar filtragens e previsões, bem como calcular a média global dos consumos elétricos registados.

Para cumprir esta tarefa, tivemos de aplicar vários dos conceitos que aprendemos nas cadeiras lecionadas neste primeiro semestre:

-**Algoritmia e Programação**: desenvolvimento da aplicação no programa *NetBeans* e do respetivo código;

-**Álgebra Linear**: definição das dimensões dos *arrays* utilizados para guardar os dados recolhidos;

- **Análise Matemática**: interpretação das respetivas fórmulas de filtragem e previsão;

-**Princípios da Computação**: utilização da linha de comandos para executar o programa (tanto em modo interativo como não interativo);

-**Laboratório e Projeto 1**: trabalho em equipa, organização e distribuição de tarefas.

Posto isto, damos como cumpridos os principais objetivos deste trabalho. Para além disso, ainda tivemos a oportunidade de aprender muito ao desenvolvê-lo: aprofundamos os conhecimentos que tínhamos em relação ao curso, otimizámos a nossa postura enquanto membros de um grupo de trabalho e aprendemos a ser mais organizados, a conseguir trabalhar sob pressão e a cumprir prazos de entrega.

Do nosso ponto de vista, o desenvolvimento do projeto correu bem, conseguimos resolver problemas e comunicar facilmente uns com os outros acerca do estado do trabalho., tendo resultado na conclusão do projeto atempadamente.

Referências

Delhij, A.; Solingen, R.; Wijnands,W. (2016, November 16). O Guia eduScrum. Retrieved January 4, 2020, from :

http://eduscrum.nl/de/file/CKFiles/O_guia_eduScrum.pdf

Manuel, J., & Xavier, N. (2016). UNIVERSIDADE ABERTA ANÁLISE E PREVISÃO DE SÉRIES TEMPORAIS COM MODELOS ARIMA E ANÁLISE ESPECTRAL SINGULAR.

Análise de Séries Temporais - 2a Edição Revista e Ampliada by Editora Blucher - Issuu. (n.d.). Retrieved January 4, 2020, from https://issuu.com/editorablucher/docs/issuu_analise_temporais_isbn9788521203896/5?e=1099747/5170484

Hugo Victor Hugo Lachos Lachos Davila, V. (n.d.). Introdução às Séries Temporais.

Exponential Moving Average - EMA Definition. (n.d.). Retrieved January 4, 2020, from https://www.investopedia.com/terms/e/ema.asp

Simple Moving Average (SMA) Definition. (n.d.). Retrieved January 4, 2020, from https://www.investopedia.com/terms/s/sma.asp

Chatfield, C. (2003). The analysis of time series: an introduction. Retrieved from https://content.taylorfrancis.com/books/download?dac=C2009-0-14379-9&isbn=9780429208706&format=googlePreviewPdf

ANEXOS

ANEXO A _ Testes Unitários

```
public void testReadFileDAYTON() throws Exception {
    System.out.println("readFile");
    int[] consumptionMW = new int[26304];
    LocalDateTime[] dateTime = new LocalDateTime[26304];
    int expResult = 22680;
    int result = LAPR1_1DK_Mafia.readFile(consumptionMW, dateTime, file);
    assertEquals(expResult, result);
}
```

Figura 14 - Teste para o método readFile com o ficheiro DAYTON.csv como input.

```
public void testDayPeriodMadrugada() throws Exception {
    System.out.println("dayPeriodMadrugada - 1 mês");
    int[] consuptionMW = {1793, 1741, 1694, 1659, 1630, 1643, 1677, 1719, 1780, 1793, 1854, 1883,/*...*/
    LocalDateTime[] dateTime = new LocalDateTime[26304];
    int startPeriod = 0;
    int size = consuptionMW.length;
    LAPR1_1DK_Mafia.dayPeriod(consuptionMW, size, startPeriod);
    size = LAPR1_1DK_Mafia.exchangeInfoDayPeriods(consuptionMW, size, startPeriod, dateTime);
    int[] expResult = {10160, 10515, 10180, 11015, 12689, 12506, 11935, 11051, 9477, 8913, 13177,/*...*/
    int[] result = Arrays.copyOf(consuptionMW, size);
    assertTrue(Arrays.equals(expResult, result));
}
```

Figura 15 – Teste para o método dayPeriod, tendo como input os valores de janeiro 2016 do ficheiro DAYTON.csv[4]

```
public void testMonthlyPeriod() throws Exception {
    System.out.println("monthlyPeriod - 1 mes completo + 1 incompleto");

    int[] auxConsumptionMW = new int[26304];
    LocalDateTime[] auxDateTime = new LocalDateTime[26304];
    LAPR1_1DK_Mafia.readFile(auxConsumptionMW, auxDateTime, file);

    int[] consumptionMW = {1793, 1741, 1694, 1659, 1630, 1643, 1677, 1719, 1780,/*...*/
    LocalDateTime[] dateTime = Arrays.copyOf(auxDateTime, 1100);
    int size = consumptionMW.length;
    LAPR1_1DK_Mafia.monthlyPeriod(consumptionMW, dateTime, size);
    int[] expResult = {1618977, 776076};
    int[] result = Arrays.copyOf(consumptionMW, 2);
    assertTrue(Arrays.equals(expResult, result));
}
```

Figura 16 - Teste para o método monthlyPeriod, tendo como input todos os valores até 15/02/2016 19:00:00 do ficheiro DAYTON.csv

[4] Foram elaborados testes para os seguintes cenários: madrugada, manhã, tarde e noite.

```java
public void testGetMonthLengthJan() {
    System.out.println("getMonthLengthJanuary");
    LocalDateTime[] dateTime = {LocalDateTime.of(2016, 1, 9, 10, 0, 0), /*...*/
    int index = 0;
    int expResult = 31;
    int result = LAPR1_1DK_Mafia.getMonthLength(dateTime, index);
    assertEquals(expResult, result);
}
```

Figura 17 - Teste para o método getMonthLength, tendo como input a data 09/01/2016.[5]

```java
public void testAnnualPeriod() throws Exception {
    System.out.println("annualPeriod - 1 ano completo + 1 incompleto");
    int[] auxConsumptionMW = new int[26304];
    LocalDateTime[] auxDateTime = new LocalDateTime[26304];
    LAPR1_1DK_Mafia.readFile(auxConsumptionMW, auxDateTime, file);

    int[] consumptionMW = Arrays.copyOf(auxConsumptionMW, 9500);
    LocalDateTime[] dateTime = Arrays.copyOf(auxDateTime, 9500);
    int size = 9500;
    LAPR1_1DK_Mafia.annualPeriod(consumptionMW, dateTime, size);
    int[] expResult = {17825259, 1492234};
    int[] result = Arrays.copyOf(consumptionMW, 2);
    assertTrue(Arrays.equals(expResult, result));
}
```

Figura 18 - Teste para o método annualPeriod, tendo como input os valores até à data 30/01/2017 19:00:00 do ficheiro DAYTON.csv

```java
public void testExchangeInfoDayPeriods() {
    System.out.println("exchangeInfoDayPeriods - 1 mês");
    int[] consumptionMW = {1793, 1741, 1694, 1659, 1630, 1643, 1677, 1719, 1780, 1793, 1854, 1883,/*...*/
    int size = consumptionMW.length;
    LocalDateTime[] dateTime = new LocalDateTime[size];
    int start = 0;
    int expResult = 31;
    int result = LAPR1_1DK_Mafia.exchangeInfoDayPeriods(consumptionMW, size, start, dateTime);
    assertEquals(expResult, result);
}
```

Figura 19 - Teste para o método exchangeInfoDayPeriods, tendo como input os valores de janeiro 2016 do ficheiro DAYTON.csv[6]

[5] Foram elaborados testes para datas pertencentes a meses com diferentes dias. No nosso caso: janeiro, fevereiro e junho.

[6] Os testes para os métodos exchangeInfoDays e exchangeInfoMonthsYears foram elaborados de forma semelhante.

```
public void testMergeSort() {
    System.out.println("mergeSort");
    int[] consumptionMW = {1793, 1741, 1694, 1659, 1630, 1643, 1677, 1719, 1780, 1793,/*...*/
    int start = 0;
    int end = consumptionMW.length - 1;
    LAPR1_1DK_Mafia.mergeSort(consumptionMW, start, end);
    int[] expResult = {1630, 1643, 1659, 1677, 1694, 1719, 1741, 1780, 1793, 1793, 1854,/*...
    int[] result = consumptionMW;
    assertTrue(Arrays.equals(expResult, result));
}
```

Figura 20 - Teste para o método mergeSort[7]

```
public void testMediaMovelSimplesAnual() throws Exception {
    System.out.println("MediaMovelSimplesAnual, n = 2");
    int[] consumptionMW = {17825259, 17295849, 10604815};
    int n = 2;
    double[] result = LAPR1_1DK_Mafia.MediaMovelSimples
    double[] expResult = {0, 17560554, 13950332};
    assertTrue(Arrays.equals(expResult, result));
}
```

Figura 21 - Teste para o método MediaMovelSimples, para 3 anos e n = 2.

```
public void testMediaMovelPesada() throws Exception {
    System.out.println("MediaMovelPesada - anual");
    int[] consumptionMW = {17825259, 17295849, 10604815};
    int size = consumptionMW.length;
    double alpha = 0.3;
    double[] result = LAPR1_1DK_Mafia.MediaMovelPesada
    double[] expResult = {17825259, 17666436, 15547949.7};
    assertTrue(Arrays.equals(expResult, result));
}
```

Figura 22 - Teste para o método MediaMovelPesada, para 3 anos e alfa = 0,3.

```
public void testAbsoluteErrorSimples() {
    System.out.println("absoluteErrorSimples");
    int[] consumptionMW = {1618977, 1451230, 1370286, 1292096, 1331203, 1548289, 1688546, 1786800, 1498560};
    double[] arrayY = {0, 0, 0, 0, 0, 0, 1471518, 1495492, 1502254};
    int size = arrayY.length;
    int result = (int)LAPR1_1DK_Mafia.absoluteError(consumptionMW, arrayY, size, out, args);
    int expResult = (int)170676.4286;
    assertTrue(result == expResult);
}
```

Figura 23 - Teste para o método absoluteError utilizando a média móvel simples.[8]

[7] O teste para o método inverseMergeSort foi elaborado com uma estrutura idêntica.

[8] Foi também elaborado um teste para o erro absoluto utilizando a média móvel pesada de forma semelhante.

```
public void previsionType2018Simples() throws Exception {
    System.out.println("previsionType - 2018 - MMS");
    int[] consumptionMW = {17825259, 17295849, 10604815};
    LocalDateTime[] dateTime = {LocalDateTime.of(2016, 01, 01, 0, 0),
        LocalDateTime.of(2017, 01, 01, 0, 0), LocalDateTime.of(2018, 01, 01, 0, 0)};
    int size = consumptionMW.length;
    int index = 1;
    int n = 2;
    double[] mediaMovelSimples = LAPR1_1DK_Mafia.MediaMovelSimples(consumptionMW, size,/*...*/
    System.out.println(Arrays.toString(mediaMovelSimples));
    double result = mediaMovelSimples[index];
    double expResult = 17560554;
    System.out.println(result);
    assertTrue(expResult == result);
}
```

Figura 24 - Teste para o método previsionType, tendo em conta a previsão do último de um conjunto de 3 anos, utilizando a média móvel simples e n = 2.[9]

```
public void testVerifyDateOutOfBounds() {
    System.out.println("verifyDateOutOfBounds");
    String inputDate = "20151231 00:00";
    LocalDateTime[] dateTime = {LocalDateTime.of(2016, 1, 1, 0, 0), LocalDateTime.of(2017, 1, 1, 0, 0)};
    int size = dateTime.length;
    int option = 5;
    LocalDateTime expResult = null;
    LocalDateTime result = LAPR1_1DK_Mafia.verifyDate(inputDate, dateTime, size, option, args, out);
    assertEquals(expResult, result);
```

```
public void testVerifyDateInvalid() {
    System.out.println("verifyDateInvalid");
    String inputDate = "20160230";
    LocalDateTime[] dateTime = {LocalDateTime.of(2016, 1, 1, 0, 0), LocalDateTime.of(2017, 1, 1, 0, 0)};
    int size = dateTime.length;
    int option = 5;
    LocalDateTime expResult = null;
    LocalDateTime result = LAPR1_1DK_Mafia.verifyDate(inputDate, dateTime, size, option, args, out);
    assertEquals(expResult, result);
}
```

Figura 25 - Testes para o método verifyDate, tendo em conta uma data inexistente no ficheiro DAYTON.csv (acima) e uma data inexistente no calendário (abaixo).

```
public void testSearchForDateIndex() {
    System.out.println("searchForDateIndex");
    LocalDateTime[] dateTime = {LocalDateTime.of(2016, 01, 01, 0, 0),
        LocalDateTime.of(2017, 01, 01, 0, 0), LocalDateTime.of(2018, 01, 01, 0, 0)};
    LocalDateTime date = LocalDateTime.of(2018, 01, 01, 0, 0);
    int option = 7;
    long expResult = 2;
    long result = LAPR1_1DK_Mafia.searchForDateIndex(dateTime, date, option, args);
    assertEquals(expResult, result);
}
```

Figura 26 - Teste para o método searchForDateIndex.

[9] Foi também elaborado um teste utilizando a média móvel pesada para prever o consumo do dia 20/01/2016 do ficheiro DAYTON.csv de forma semelhante. Adicionalmente foi testada a previsão para a primeira observação de um array (sem valores anteriores com os quais seja possível efetuar cálculos).

ANEXO B _ Diagramas

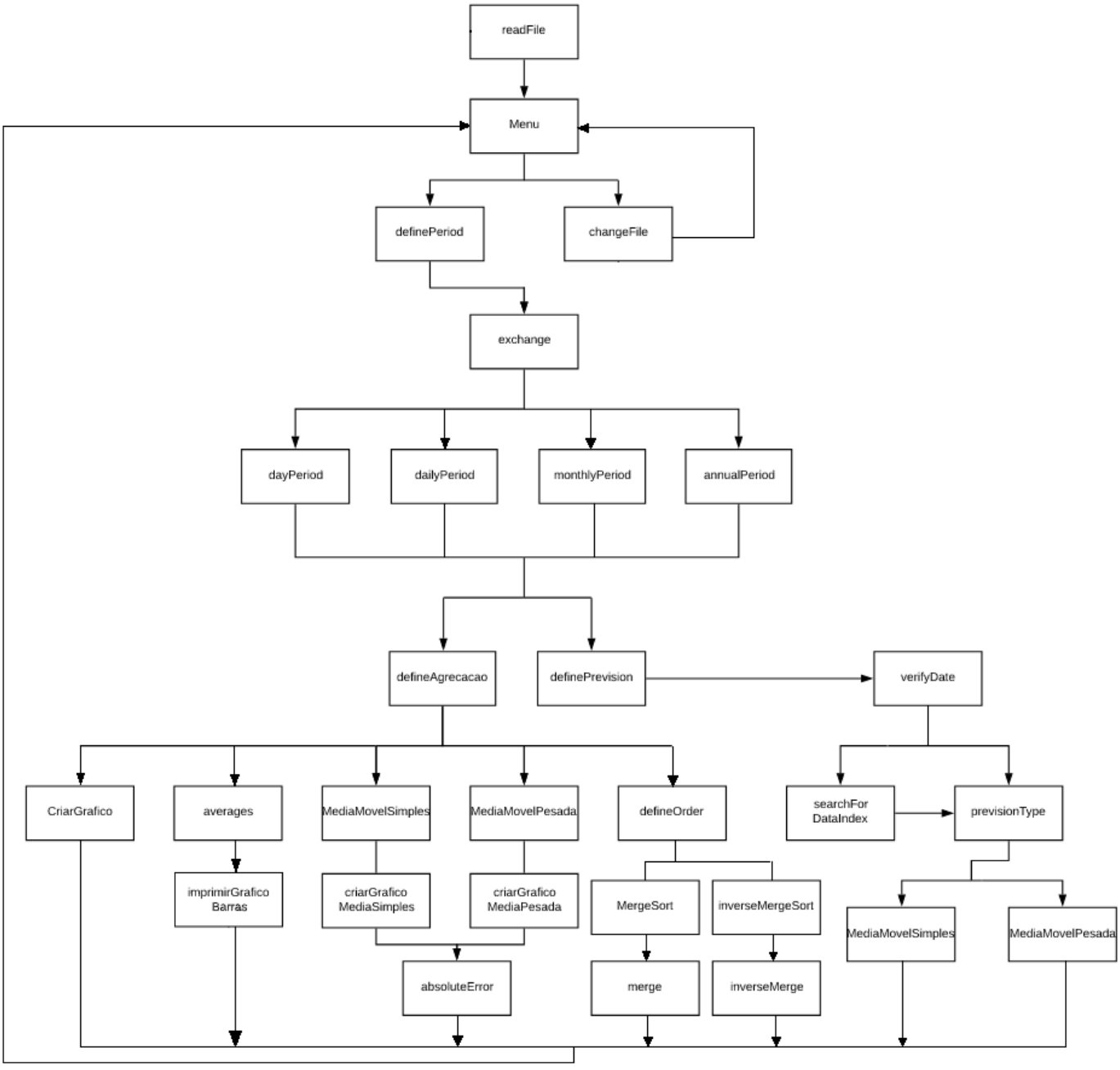

Figura 27 - Modo Interativo

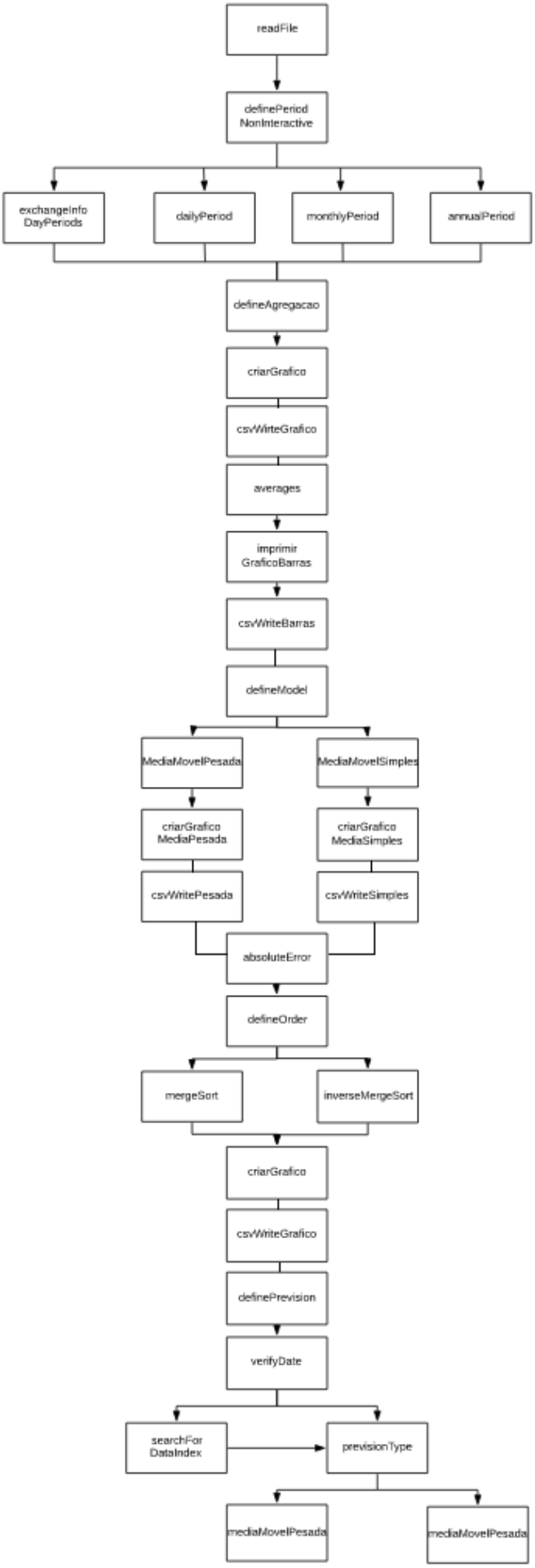

Figura 28 - Modo Não Interativo