DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

Relatório do Projeto de LAPR1

23 de janeiro de 2022

Turma, número da equipa

1191296, Gabriel Gonçalves

1191369, Tiago Leite

1211314, João Durães

1211304, Francisco Bogalho

Ana Barata (ABT)

Carlos Ferreira (CGF)

Stella Abreu (SAU)

isep Instituto Superior de Engenharia do Porto

Ìndice

1. Introdução

O presente relatório tem como objetivo descrever toda a informação relevante sobre o projeto desenvolvido, no âmbito da unidade curricular Laboratório/Projeto I da Licenciatura em Engenharia Informática do Instituto Superior do Porto.

O objetivo do trabalho consiste na consolidação e extensão dos conhecimentos de todas as unidades curriculares que funcionaram no primeiro semestre aplicando o processo básico de desenvolvimento de aplicações informáticas e valorizando todas as suas fases desde a análise e conceção aos testes de validação.

Este documento encontra-se dividido em seis capítulos, que se encontram repartidos em secções e subsecções com o objetivo de apresentar ao detalhe o projeto desenvolvido.

Neste presente capítulo é realizada uma contextualização ao trabalho realizado, sendo apresentado os seus objetivos e a estrutura deste documento.

No segundo capítulo é apresentada a metodologia de trabalho, sendo abordada a gestão de projetos *Scrum*, o planeamento e distribuição de tarefas e uma reflexão crítica sobre a dinâmica da equipa.

No terceiro capítulo encontra-se um estudo realizado sobre as cadeias de *Markov* e a decomposição *LU* pelo método de *Crout*, incluindo exemplos ilustrativos da sua aplicação.

O quarto capítulo é composto pela apresentação da organização dos módulos do projeto e os métodos principais utilizados.

No quinto capítulo é realizado um caso de estudo, sendo realizado uma análise e discussão global dos resultados obtidos.

No último capítulo é apresentada uma conclusão do projeto, sendo realizado um resumo aos pontos principais explicitados no corpo central do relatório, também realçado os principais resultados alcançados e uma reflexão sobre o trabalho desenvolvido, apontando sugestões de aplicabilidade e melhoria.

2. Metodologia de Trabalho

Nesta secção aborda-se a metodologia de trabalho, em particular a abordagem de gestão de projetos *Scrum* e como esta foi aplicada ao logo do desenvolvimento do trabalho. Também é apresentado como foi elaborado o planeamento de trabalho e a organização em equipa, incluindo todas as ferramentas utilizadas e a sua utilidade. Por fim é realizada uma reflexão crítica sobre a dinâmica da equipa.

2.1 Scrum no desenvolvimento do Projeto

A metodologia de trabalho utilizada é o *Scrum*. O *Scrum* é uma metodologia de trabalho que promove o trabalho em equipa. O *Scrum* procura encorajar as equipas a ganharem conhecimentos através de experiências, que se auto-organizem enquanto trabalham num problema e que reflitam sobre as suas vitórias e sobre as derrotas e que melhorem continuamente.

2.2 Planeamento e distribuição de tarefas

Ao longo do desenvolvimento do trabalho, a equipa organiza-se de forma a que todos os membros possam comparecer às reuniões sem que sejam prejudicados noutras tarefas não relacionadas com este trabalho.

Todos os membros da equipa têm oportunidade de dar a sua opinião sem serem interrompidos, sendo que todas as opiniões são tomadas em conta. Posteriormente, em conjunto decidimos qual é a ideia que melhor se integrará no projeto no nosso ponto de vista.

Na tomada de decisão, o grupo reúne todas as ideias dadas por cada membro e realiza uma votação para que todos possam eleger aquela ideia que acham que será a mais adequada para o projeto. Caso a votação não seja unânime, o *Scrum Master* realiza uma triagem das melhores ideias e é feita uma votação apenas para aquelas que foram selecionadas

Quando a equipa decide realizar uma reunião, em caso de algum elemento não participar, deve justificar o porquê da sua ausência. Caso não o faça, não tem o direito de criticar as decisões tomadas pelos restantes membros.

Mesmo justificando a falta, deverá procurar manter-se a par do progresso do grupo. Se as faltas se repetirem continuamente até um certo limite e sem justificação aplausível, o membro é convidado a sair da equipa.

Para a comunicação entre os membros da equipa, para além do formato presencial, utiliza-se o *Microsoft Teams* para reuniões mais formais e o *Discord* para reuniões menos formais, troca de ideias e discussões de tópicos.

Durante o desenvolvimento de um projeto, é expectável que os membros da equipa cooperem uns com os outros ativamente, para que nenhum membro fique para trás e para que nenhum conteúdo fique esquecido. É ainda esperado também que todos os membros se preparem para as reuniões com informação pertinente, pois todos têm maturidade suficiente para assumir esta responsabilidade.

Durante o planeamento e distribuição de tarefas definiu-se a utilização de algumas ferramentas como o *Trello*, *Bitbucket* & *SourceTree*, *Microsoft Teams*, *Discord* e *IntelliJ IDEA*. Na Tabela 1 realiza-se uma comparação entre estas ferramentas, explorando em cada uma delas o seu objetivo, utilização, vantagens e desvantagens.

2.3 Reflexão crítica sobre a dinâmica da equipa

Após uma reflexão em conjunto, todos os elementos da equipa concordaram que houve uma excelente dedicação ao trabalho que resultou numa aplicação bem planeada e concebida. O trabalho em equipa e a alta proficiência entre os membros resultou numa boa dinâmica e numa eficácia idílica na resolução de problemas complexos.

As várias etapas do trabalho foram distribuídas homogeneamente pelos quatro elementos e consoante as capacidades e dificuldades de cada um. O apoio mútuo fez com que cada um dos elementos do grupo enfrenta-se as dificuldades e dúvidas que surgiram na realização da aplicação e as esclarecesse. Criando assim uma relação de simbiose que nos incentiva a melhorar para estarmos á altura uns dos outros.

Ferramenta	Objetivo	Como utilizámos	Vantagens	Desvantagens
Trello	Organizar, dividir ideias/tarefas do desenvolvimento do projeto.	Utilizado extensivamente, pois sempre que existiam novas tarefas a implementar no projeto, eram logo colocadas no *Trello* e distribuídas pelos membros da equipa.	Didático; Intuitivo; Dinâmico.	Não é possível de aceder sem ligação à internet.
Bitbucket & SourceTree	Controlar todas as versões da aplicação.	Utilizado para fazer o controlo das versões da aplicação. Possibilita que o processo de junção de todas as partes do código seja bastante mais fácil.	Facilidade nas alterações do código em grupo.	Problemas no início da utilização.
Microsoft Teams	Comunicação entre os membros da equipa e armazenar documentos desenvolvidos.	Plataforma utilizada para guardar todos os documentos referentes ao trabalho desenvolvido pela equipa e também para realizar reuniões mais formais.	Ferramenta de comunicação bastante completa. Permite a partilha de ecrã com acesso remoto dos outros utilizadores e permite aceder a ferramentas externas.	Dificuldade no início da sua utilização.
Discord	Realizar a comunicação menos formal entre os membros da equipa.	Utilizado para realizar a comunicação entre todos os membros da equipa de uma forma um pouco menos formal e onde foram discutidas algumas decisões a tomar.	Vários membros podem fazer partilha de ecrã, sem que as outras sejam paradas; Facilidade de utilização.	As partilhas de ecrã apenas têm resolução 720p; Utiliza mais recursos do hardware do que o Teams.
IntelliJ IDEA	Desenvolver software de computador na linguagem *JAVA*.	Utilizamos para o desenvolvimento e programação da aplicação na linguagem *JAVA*.	Fácil utilização; Muito fácil compreender erros e corrigi-los; Existência de imensos plugins que facilitam a utilização.	Elevado gasto de recursos.

Tabela 1 - Comparação das ferramentas utilizadas

3. Análise de cadeias de Markov e decomposição LU pelo método de Crout

3.1 Cadeias de Markov

As cadeias de *Markov* são processos aleatórios que satisfazem a propriedade de *Markov*. Defendida por *Andrey Markov*, talvez o mais famoso matemático russo do final do século XIX. É processo aleatório que satisfaz a propriedade *Markov* todo aquele que possui a seguinte característica. Pode-se fazer previsões para o futuro somente com base no seu estado atual independentemente do que aconteceu no passado. Como tal, as cadeias de *Markov* são bastante gerais e desfrutam de uma ampla gama de aplicações. Por exemplo, na física, são amplamente utilizados na mecânica estatística. Nas ciências da informação, é usada no processamento de sinais, codificação, compactação de dados e reconhecimento de padrões. E nos estudos sociais é usada na previsão da evolução e propagação de pandemias e de outras doenças infeciosas. (Markov Chains, 2018)

As cadeias de *Markov* têm por base o uso das matrizes de transição para prever os valores. Estas consistem numa matriz de ordem igual á da matriz com que estamos a trabalhar e servem para prever os valores para situações que ainda não aconteceram.

As matrizes de transição são na essência uma função que recebe um valor por cima e transforma-o com a probabilidade do que pode ser como mostra a Figura 1, por exemplo se o valor de A for vinte e de B for dez significa que depois da transformação A terá o valor de quinze e B o valor de quinze.

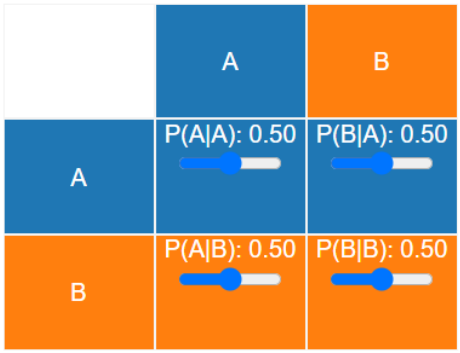

Figura 1 - Exemplo do funcionamento da matriz de transição

Com o objetivo de prever os valores de uma pandemia para determinada data precisamos de aplicar uma matriz de transição a um grupo de dados adequado. Mas para conseguir aplicar as matrizes de transição primeiro devemos elevar a matriz de transição á quantidade de dias no futuro que queremos prever os valores, ou seja, por exemplo se quisermos prever os valores do dia 7/04/2022 e os dados acabarem dia 31/03/2022 então temos de elevar a matriz de transição a sete, mas se quisermos calcular os valores do dia seguinte elevamos a matriz a um o que não a altera.

Para chegar ao valor que queremos prever precisamos multiplicar os valores de determinada célula pelos valores que temos da coluna, da matriz de transição já alterada, correspondente e fazer o mesmo processo para todas as colunas, de seguida somar os valores resultantes das linhas.

07/01/2022 ⟶	9893098	254240	1353	161	17
	Não Infetados	Infetados	Hospitalizados	Internado UCI	Obitos
Não Infetados	0.9995	0.003	0.002	0.001	0
Infetados	0.0005	0.96	0.004	0.015	0
Hospitalizados	0	0.007	0.98	0.02	0
Internado UCI	0	0.003	0.01	0.95	0
Obitos	0	0	0.004	0.014	1

Figura 2 - Exemplo da aplicação da matriz de transição aos dados

O resultado obtido a partir dos cálculos anteriormente apresentados representa os valores que estão previstos para a data que foi calculada.

No contexto de prever os valores de determinado dia usamos um método para imprimir os valores, um para guardá-los e uma para chamar os métodos necessários para construir os valores em si.

No método que cria os valores é chamado um método para alterar a matriz de transição para se poder através da mesma obter os valores, do dia pretendido.

3.2 Decomposição LU pelo método de *Crout*

A decomposição LU pelo método de *Crout* é aconselhada para programas que utilizem tais funcionalidades matemáticas pois facilita a criação do código na medida em que a realização da inversa de uma matriz triangular inferior ou superior é substancialmente menos exigente computacionalmente.

De modo geral para alcançarmos a inversa pelo método de Crout temos de decompor uma matriz pela decomposição LU e depois inverter L pelo método de substituição para trás e repetir para U, depois de ter L e U invertidos multiplicamo-los. (William H. Press, 2002)

Para realizar uma decomposição em LU pelo método de Crout o primeiro passo é aplicar a seguinte formula: $\beta_{ij} = a_{ij} - \sum_{k=1}^{i-1} \alpha_{ik}\beta_{kj}$, em que as matrizes $\alpha_{ik}\beta_{kj}$ são da matriz que estamos a tentar decompor e a_{ij} é uma posição, de modo a termos a matriz triangular superior dada por β, de seguida aplicaremos: $\alpha_{ij} = \frac{1}{\beta_{jj}}\left(a_{ij} - \sum_{k=1}^{j-1} \alpha_{ik}\beta_{kj}\right)$, em que as matrizes $\alpha_{ik}\beta_{kj}$ são da matriz que estamos a tentar decompor e a_{ij} é uma posição , para obtermos a matriz triangular inferior dada por α.

Este método recebe uma matriz para ser decomposta em LU e altera diretamente nas variáveis do método em que é chamado.

$$\beta_{ij} = a_{ij} - \sum_{k=1}^{i-1} \alpha_{ik}\beta_{kj}$$

O ciclo de iteração ilustrado na Figura 3 *representa* .

```
//Calcula a matriz triangular superior
for (int i = 0; i < j; i++) {
    sum = matrixToDecompose[i][j];
    for (int k = 0; k < i; k++) {
        sum -= matrixToDecompose[i][k] * matrixToDecompose[k][j];
    }
    matrixToDecompose[i][j] = sum;
}
```

Figura 3 - Cálculo da matriz triangular superior

O ciclo de iteração apresentado Figura 4 representa .

$$\alpha_{ij} = \frac{1}{\beta_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} \alpha_{ik}\beta_{kj} \right)$$

```
//Calcula a matriz triangular inferior
for (int i = j; i < n; i++) {
    sum = matrixToDecompose[i][j];
    for (int k = 0; k < j; k++) {
        sum -= matrixToDecompose[i][k] * matrixToDecompose[k][j];
    }
    matrixToDecompose[i][j] = sum;
    dum = vv[i] * Math.abs(sum);
    if (dum >= big) {
        big = dum;
        imax = i;
    }
}
```

Figura 4 - Cálculo da matriz triangular inferior

Para aplicar o método de substituição para trás utilizar-se-á a fórmula A*A ^-1 = I, em que A representa a matriz que queremos inverter, A^-1, sendo uma matriz só com incógnitas, é o resultado que queremos descobrir e I é a matriz identidade de A. O primeiro passo é multiplicarmos A por A^-1 e de seguida igualar a matriz resultante aos respetivos valores na matriz identidade, obtendo assim um sistema de, no caso de A ser uma matriz quatro por quatro, dezasseis equações.

Este método receberá a matriz, a ordem da matriz e os resultados das equações para poder operar. Ele não devolvera um valor pois efetua as alterações diretamente nas variáveis do método onde é chamado.

O passo final será multiplicar o inverso da matriz triangular superior pelo inverso da matriz triangular inferior, ambas calculadas nos passos anteriores, obtendo assim a inversa da matriz inicial.

A nível de código, o método recebe uma matriz para ser invertida pela decomposição em *LU* e chamará os métodos necessários (*luDecomposition*, *backwardsSubstitution*) e devolverá a matriz invertida como mostra a Figura 5.

```
//Inverte a matrix
public static double[][] inverseMatrix(double[][] matrixToInvert) {

    int i, j;
    double[][] initialArray = copyMatrix(matrixToInvert, matrixToInvert.length);
    int length = matrixToInvert.length;
    double[] col = new double[length];

    double[][] inverseMatrix = new double[length][length];

    double trades = luDecomposition(initialArray);

    for (j = 0; j < length; j++) {
        for (i = 0; i < length; i++) {
            col[i] = 0.0;
        }

        col[j] = 1.0;
        backwardsSubstitution(initialArray, length, col);

        for (i = 0; i < length; i++) {
            inverseMatrix[i][j] = col[i];
        }
    }

    return inverseMatrix;
}
```

Figura 5 - Inversa da matriz

Em ordem a conseguirmos prever a quantidade de dias que se espera que um individuo demora a passar do seu estado atual (não infetado, infetado, hospitalizado, internado UCI) para o estado absorvente, o óbito, tem que se seguir alguns passos matemáticos.

O primeiro passo será subtrair a matriz de transição sem o estado absorvente á matriz identidade da mesma. Para se descobrir a matriz de transição sem o estado temos que cortar a última coluna e linha da matriz de transição original.

O segundo passo será inverter a subtração resultante do último passo pelo método de decomposição em LU.

O passo final é somar todos os valores das colunas obtendo assim a quantidade de dias que uma individuo demora a passar do seu estado atual para o estado de óbito.

Para a criação do código optamos por utilizar um método, *theFinalCountdown* para chamar todos os métodos necessários como mostra a Figura 6.

```java
//Calcula a quantidade esperada de dias que um individuo demora a morrer
public static double[] theFinalCountdown(double[][] transitionMatrix) {
    double[][] transitionMatrixWithoutAbsorbingState = createTransitionMatrixWithoutAbsorbingState(transitionMatrix);
    double[][] matrixOfDaysUntilDeath;
    double[][] identityMatrix = createIdentityMatrix(transitionMatrixWithoutAbsorbingState.length);
    double[][] subtractedMatrix;
    double[] amountOfDaysUntilDeath;

    //Primeiro passo da fórmula - subtrair a matriz de transição sem o estado absorvente à matriz de identidade
    subtractedMatrix = subtractTwoMatrices(identityMatrix, transitionMatrixWithoutAbsorbingState);

    //Segundo passo da fórmula - inverter o resultado da subtração
    matrixOfDaysUntilDeath = inverseMatrix(subtractedMatrix);

    // Soma de todas as colunas da matriz
    amountOfDaysUntilDeath = sumOfMatrixColumns(matrixOfDaysUntilDeath, matrixOfDaysUntilDeath.length);

    return amountOfDaysUntilDeath;
}
```

Figura 6 - Calcula a quantidade esperada de dias que um indivíduo demora a morrer

Sendo o primeiro o subtractedTwoMatrix. Este método recebe duas matrizes, de qualquer ordem, e subtrai a segunda á primeira, desde que ambas sejam da mesma ordem, devolvendo uma só matriz como ilustrado na Figura 7 realizando assim o primeiro passo.

```java
//Subtrai uma matriz a outra matriz desde que sejam da mesma ordem
public static double[][] subtractTwoMatrices(double[][] minuedMatrix, double[][] subtrahendMatrix) {
    double[][] differenceMatrix = new double[subtrahendMatrix.length][subtrahendMatrix[0].length];

    for (int i = 0; i < subtrahendMatrix.length; i++) {
        for (int j = 0; j < subtrahendMatrix[0].length; j++) {
            differenceMatrix[i][j] = minuedMatrix[i][j] - subtrahendMatrix[i][j];
        }
    }

    return differenceMatrix;
}
```

Figura 7 - Método para subtrair duas matrizes

Para o segundo passo utilizamos o método de calcular a inversa pela decomposição em LU que foi previamente explicado.

No terceiro passo apenas terá de ser chamado o método *sumOfMatrixColumns*. Este método soma todos os valores das colunas da matriz como mostra a Figura 8 resultante do último passo.

```java
//Soma os valores das colunas por linha
public static double[] sumOfMatrixColumns(double[][] matrix, int numberMatrixColumns) {
    double[] sumOfMatrixColumns = new double[numberMatrixColumns];

    for (int column = 0; column < numberMatrixColumns; column++) {
        for (int line = 0; line < matrix[column].length; line++) {
            sumOfMatrixColumns[column] += matrix[line][column];
        }
    }

    return sumOfMatrixColumns;
}
```

Figura 8 - Método para somar todos os valores das colunas por linhas

4. Desenvolvimento e Implementação da Aplicação

Nesta secção aborda-se o desenvolvimento e implementação da aplicação, em particular a organização dos módulos do projeto, sendo também explicado o funcionamento da aplicação no modo interativo e não interativo e a implementação dos métodos mais importantes.

4.1 Organização dos módulos do projeto

Para uma melhor perceção da organização dos módulos do projeto realizou-se um diagrama que se encontra na Figura 28.

4.2 Explicação do funcionamento da aplicação no modo interativo

Ao iniciar a aplicação esta apresenta um menu principal apresentado na Figura 9**Erro! A origem da referência não foi encontrada.** que é constituído por 5 opções. Entre as várias opções é possível carregar ficheiros, realizar a análise de diferentes resoluções temporais, a análise comparativa ou prever a evolução da pandemia, todas estas opções têm associado um sub-menu. Existem também uma opção que permite ao utilizador terminar o programa.

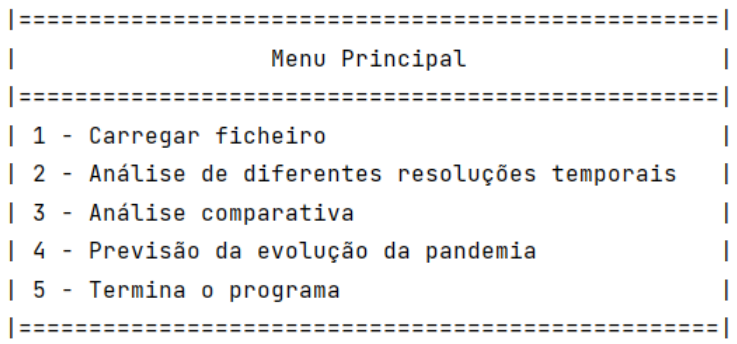

Figura 9 - Menu Principal

No sub-menu de importar ficheiros é possível carregar ficheiros com os dados acumulados, totais ou com a matriz de transição como ilustra a Figura 10.

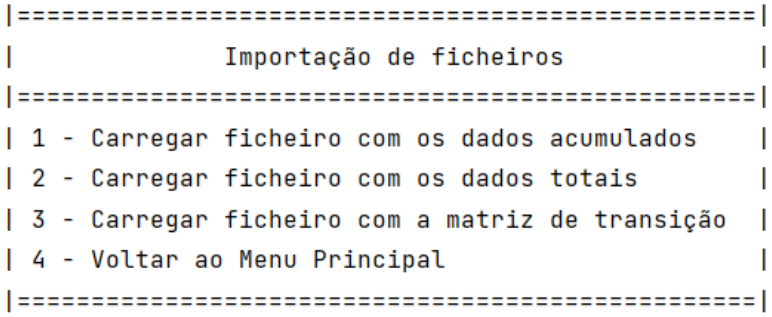

Figura 10 - Menu para a importação de ficheiros

Ao selecionar cada uma das opções o utilizador deve inserir o nome do ficheiro incluindo a sua extensão e caso não se encontre no diretório atual o seu caminho. Seguidamente a aplicação valida se o ficheiro existe e contém a extensão pretendida, pois no caso do ficheiro com os dados acumulados e totais este deve ser ".csv" e na matriz de transição ".txt". Se tudo estiver correto o ficheiro é carregado com sucesso.

Ao selecionar a opção de análise de diferentes resoluções temporais ou de análise comparativa o utilizador é direcionado para um sub-menu, onde deve escolher se pretende visualizar os novos casos ou os dados totais como ilustra a Figura 12.

```
|================================================|
|              Tipo de Visualização              |
|================================================|
| 1 - Visualizar a análise dos novos casos Covid |
| 2 - Visualizar a análise dos dados totais Covid|
| 3 - Voltar ao Menu Principal                   |
|================================================|
```

Figura 12 - Menu para o tipo de visualização

Após escolher uma das opções, a aplicação apenas permite avançar se já existiu o carregamento do ficheiro que contém os respetivos dados que se pretende visualizar.

Em relação à análise de diferentes resoluções temporais existe ainda mais um sub-menu, onde o utilizador deve indicar se prende visualizar a análise diária, semanal ou mensal como mostra a Figura 13.

```
|==============================|
|       Análise temporal       |
|==============================|
| 1 - Análise Diária           |
| 2 - Analise Semanal          |
| 3 - Análise Mensal           |
| 4 - Voltar ao Menu Anterior  |
|==============================|
```

Figura 13 - Menu para o tipo de análise temporal

Após escolher o tipo de análise temporal é pedido um intervalo de datas ao utilizador, podendo estas serem escritas no formato DD-MM-AAAA ou AAAA-MM-DD. Após a aplicação validar o intervalo de datas inserido, caso este esteja correto reencaminha o utilizador para um sub-menu que permite escolher qual o tipo de dados a visualizar como mostra a Figura 14. Após o utilizador escolher a opção pretendida, se existir dados nesse intervalo estes serão mostrados em formato de tabela, podendo o utilizador se pretender guardá-los num ficheiro.

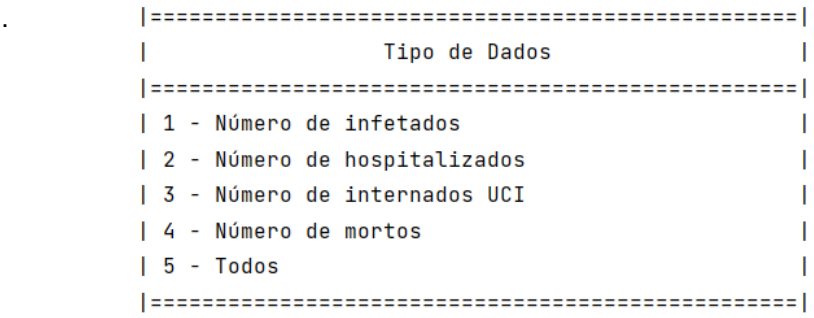

Figura 14 - Menu para o tipo de dados

Relativamente à análise comparativa o processo é muito semelhante ao apresentado anteriormente sendo a maior diferença o pedido de dois intervalos de datas ao utilizador.

No que diz respeito à opção de prever a evolução da pandemia, esta é redirecionada para um sub-menu que apenas está disponível se anteriormente foi carregado um ficheiro com a matriz de transição. Na previsão de dados é possível prever a quantidade de dias que um indivíduo demorar a chegar até à morte a partir dos diferentes estados e prever os dados para um determinado dia como mostra a Figura 15.

Nesta última opção é necessário o ficheiro dos dados totais estar carregado, além de ser necessário indicar a data do dia a prever. Em cada uma das opções é possível guardar os dados visualizados num ficheiro.

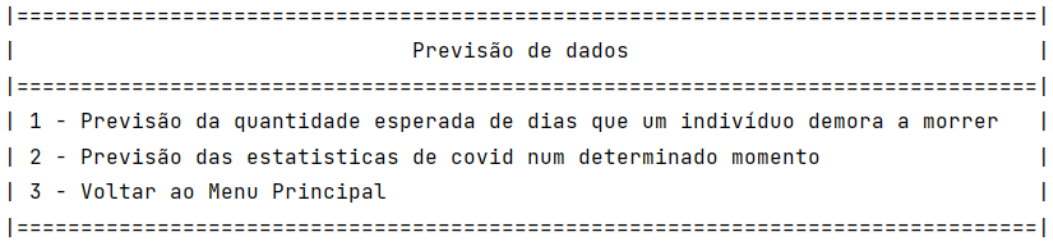

Figura 15 - Menu para a previsão de dados

4.3 Explicação do funcionamento da aplicação no modo não interativo

No modo não interativo existem 3 opções que podem ser executadas pelo utilizador.

Para realizar apenas a previsão dos dados e estimar o número médio de dias até à morte, o comando a executar deve conter além da data a prever, a identificação do ficheiro com os dados acumulados, a matriz de transição e o ficheiro onde é guardada toda a informação apresentada. Um exemplo de comando para esta opção é:

java -jar Covid19Analysis.jar -T 08-01-2022 exemploRegistoNumeroTotais.csv exemploMatrizTransicoes.txt FicheiroSaida1.txt

Para não realizar as duas funcionalidades anteriores e realizar a análise a diferentes resoluções temporais (diária, semanal ou mensal) e a análise comparativa deve ser indicado as datas para cada uma destas funcionalidades, o ficheiro com os dados acumulados e o nome do ficheiro onde serão guardados os dados apresentados. Para esta opção um exemplo de comando é:

java -jar Covid19Analysis.jar -r 1 -di 01-05-2020 -df 31-05-2020 -di1 10-04-2020 -df1 20-04-2020 -di2 10-05-2020 -df2 20-05-2020 exemploRegistoNumerosAcumulados.csv FicheiroSaida2.txt

Para realizar todas as funcionalidades do programa, o comando deve possuir todos os argumentos necessários desde as datas, a identificação do ficheiro dos dados totais, acumulados e a matriz de transição. Como em todas as opções anteriores também deve ser indicado o nome do ficheiro onde é pretendido guardar a informação apresentada. Para esta opção uma amostra de comando é:

java -jar Covid19Analysis.jar -r 0 -di 01-05-2020 -df 10-05-2020 -di1 10-04-2020 -df1 20-04-2020 -di2 10-05-2020 -df2 20-05-2020 -T 08-01-2022 exemploRegistoNumeroTotais.csv exemploRegistoNumerosAcumulados.csv exemploMatrizTransicoes.txt FicheiroSaida3.txt

4.4 Explicação da implementação da aplicação

Devido ao requerimento do cliente em possuir um modo interativo e um modo não interativo, optou-se por no método *main* "separar" o caminho que a aplicação vai percorrer, através da análise do tamanho dos argumentos como mostra a Figura 16.

```
if (args.length == 0) {
    mainMenu(covidAccumulatedDailyData, covidTotalDailyData, countAccumulatedDataRecords, countTotalDataRecords);
} else {
    nonInteractiveMode(args, covidAccumulatedDailyData, covidTotalDailyData);
}
```

Figura 16 - Código para escolha do caminho a percorrer pela aplicação

No caso de não existirem, ou seja, serem iguais a *0* (zero), o programa é direcionado para um método que contém um menu para existir interação por parte do utilizador, senão é encaminhado para um método que analisa o número de argumentos passados e caso estes estejam corretos é redirecionado como mostra a Figura 17 para o respetivo método que valida os argumentos e executa as respetivas funcionalidades.

Optou-se por dividir as opções do modo interativo em 3 devido a cada comando possuir um número de argumentos e funcionalidades diferentes.

```
if (args.length == NUMBER_ARGUMENTS_OPTION1) {
    nonInteractiveModeOption1(args, covidTotalDailyData);
} else if (args.length == NUMBER_ARGUMENTS_OPTION2) {
    nonInteractiveModeOption2(args, covidAccumulatedDailyData);
} else if (args.length == NUMBER_ARGUMENTS_OPTION3) {
    nonInteractiveModeOption3(args, covidAccumulatedDailyData, covidTotalDailyData);
} else {
    System.out.println("Erro! Comando inválido! Exemplo: java -jar nome programa.jar -r X -di DD-MM-AAAA -df DD-MM-AAAA " +
            "-di1 DD-MMAAAA -df1 DD-MM-AAAA -di2 DD-MM-AAAA -df2 DD-MM-AAAA -T DD-MM-AAAA " +
            "registoNumeroTotalCovid19.csv registoNumerosAcumuladosCovid19.csv matrizTransicao.txt nomeficheirosaida.txt");
}
```

Figura 17 - Código para validar o número de argumentos passados

Para a leitura dos ficheiros com dados *Covid-19*, optou-se por criar um método que preenche um *array* bidimensional passado por referência e que devolve o número de registos que foram guardados no *array* (Figura 18). Assim, com este funcionamento, não é necessário criar um novo método apenas para contar quantos registos estão inseridos no *array*, tornando a aplicação mais eficiente.

```
// Fazer a leitura dos dados do ficheiro
public static int readDataFile(int[][] diaryCovidData, String fileName) throws FileNotFoundException {

    int amountOfRecordedData = 0;
```

Figura 18 - Cabeçalho do método que realiza a leitura de dados Covid-19 de um ficheiro

Inicialmente, este método iria acrescentar sempre dados ao final do *array* bidimensional caso fossem introduzidos novos dados a partir de um ficheiro, mas durante uma reunião com o cliente foi-lhe questionado qual seria o comportamento que ele gostava que esta operação tivesse. O cliente, optou por querer descartar todos os dados que existiam anteriormente e apenas ficar com os que vinham no ficheiro atual.

Os dados são armazenados no array bidimensional como números inteiros, onde existem 8 posições para armazenar diferentes dados. Na posição 0 (zero) encontra-se o ano, na posição 1 o mês, na posição 2 o dia, na posição 3 os não infetados, na posição 4 os infetados, na posição 5 os hospitalizados, na posição 6 os internados em UCI, e na posição 7 as mortes. Para possibilitar uma melhor escrita do código e também para tornar o código responsivo a alterações, onde pudessem começar a existir mais tipo de estatísticas, foram criadas constantes para cada tipo de dado (

Figura 24). A equipa decidiu utilizar um *array* bidimensional de inteiros e não de *strings*, devido a ser mais fácil de trabalhar com este tipo de dados ao realizar os cálculos, não precisando assim de estar constantemente a passar uma *string* para um inteiro.

Na funcionalidade de compreensão de dados, existe a possibilidade de esta ser feita em diferentes resoluções temporais (análise diária, semanal ou mensal). Para cada uma destas resoluções o utilizador tem de indicar uma data de início e uma data de fim de um período temporal.

Para uma melhor organização do código decidiu-se criar um método que retorna um *array* com os dados compreendidos entre as duas datas. Caso os dados a analisar sejam acumulados é também realizado um cálculo entre o dia anterior e o próprio para se chegar ao valor dos novos casos.

Para a análise diária, serão apresentados todos os dados de cada dia desse mesmo período inserido. Já para a análise semanal, serão apresentados os dados de todas as semanas que se encontram completas nesse período inserido, ou seja, uma semana só é considerada se contiver dados de segunda-feira a domingo. E para a análise mensal, são apresentados os dados de todos os meses que se encontrem completos dentro do período fornecido, ou seja, que tenham dados de dia 1 do mês até ao último dia desse mesmo mês.

Nota: Na análise dos ficheiros que tem os dados dos acumulados, o primeiro dia que é legível para apresentação é o segundo dia com dados registados no ficheiro, pois apenas nesse dia é possível realizar os cálculos necessário.

Na realização da análise semanal, fazemos primeiramente a verificação de se existem semanas completas com dados no intervalo inserido pelo utilizador. Se não existirem dados de uma semana completa, é apresentada uma mensagem ao utilizador a notificar o ocorrido, caso exista faze-mos o processo explicado a seguir. Para sabermos quais seriam os dados que deveriam ser apresentados, começamos por descobrir a data de início da primeira segunda-feira no intervalo inserido pelo utilizador, em que existem dados. Após isto, descobrimos a data de fim que será a data do último domingo com dados, no intervalo inserido pelo utilizador. Após descobrir estas duas datas, partimos o *array* bidimensional desde o índice da data inicial encontrada, até à data final encontrada. Para realizar o cálculo de cada semana é feita a soma dos dados de cada dia pertencente a uma dada semana. Estes dados são armazenados num *array* bidimensional que armazena as soma de cada um dos dados de uma semana, sendo que sabemos que cada semana é de 7 em 7 dias, o que faz assim o ciclo de segunda-feira a domingo.

```
// Identificar qual é o dia da semana de uma certa data
public static int identifyWeekDay(String date) throws ParseException {
    Calendar calendar = Calendar.getInstance();
    calendar.setTime(FORMAT_DATE.parse(date));

    return calendar.get(Calendar.DAY_OF_WEEK);
}
```

Figura 19 - Método que identifica o dia da semana de uma certa data

Para descobrir a primeira segunda-feira dos dados existentes no *array* bidimensional, é realizada a verificação de cada dia existente nesse intervalo até encontrar a primeira segunda-feira.

Para descobrir o último domingo do intervalo, é depois realizado um ciclo que anda de 7 em 7 dias, sendo que isto ira obter o índice do *array* com o último registo de um domingo. O ciclo termina quando o índice do possível próximo domingo ultrapassa o número total de registos existentes.

Na realização da análise mensal, realizamos um processo muito semelhante ao aplicado para realizar a análise semanal, como a diferença que procuramos por meses completos e não por semanas completas. Nesta parte, foram desenvolvidos métodos necessários para saber quantos dias iria ter um certo mês, pois fevereiro pode ter 28 ou 29 dias. Realizou-se então um método para saber se um ano é bissexto e outro para devolver o número de dias de um determinado mês num dado ano.

```
// Método para saber se o ano é bissexto
public static boolean isLeapYear(int year) {
    return year % 400 == 0 || (year % 4 == 0 && year % 100 != 0);
}
```

Figura 20 - Método para saber se um determinado ano é bissexto

```java
// Método para saber quantos dias tem o mês
public static int howManyDaysTheMonthHave(int year, int month) {
    int[] daysOfEachMonth = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
    int[] daysOfEachMonthLeap = {31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

    if (month < 1 || month > 12) {
        return 0;
    }

    return isLeapYear(year) ? daysOfEachMonthLeap[month - 1] : daysOfEachMonth[month - 1];
}
```

Figura 21 - Método para saber quantos dias tem um mês de um determinado ano

Com a elaboração destes dois métodos torna-se possível saber quantos dias deve ou não ter um determinado mês de um dado ano. Isto faz com que a aplicação seja capaz de descobrir facilmente o índice do *array* bidimensional com um mês completo, e até mesmo descobrir o possível índice de um próximo mês (Figura 26). Para descobrir se existe dados de um determinado mês completo, basta verificar se o índice atual do *array* corresponde ao dia um dos mês. Depois disso, calculando o índice em que deve estar o último dia do mês, é só verificar se este é menor do que o tamanho de dados armazenados em memória.

Para descobrir a data do último mês completo, basta verificarmos se existem dados para todos os dias do mês seguinte ao primeiro que se encontra completo. Caso exista, é repetido este processo até chegar ao mês em que se irá verificar que o próximo já não contém dados para todos os seus dias.

Após obtermos o *array* com os dados que necessitamos para realizar a análise semanal ou mensal, calculámos os valores para cada uma das semanas ou meses completos existentes no intervalo pedido pelo utilizador (Figura 27). Torna-se assim possível fazer a apresentação dos valores obtidos ao utilizador.

Sempre que é realizado o output ao utilizador sobre alguma informação de dados e/ou estatísticas, a aplicação dá a possibilidade ao utilizador de guardar este mesmo output para um ficheiro com um nome ao seu gosto.

Os testes unitários são extremamente necessários e úteis, pois sempre que existir a alteração a um método ou a uma funcionalidade, é possível verificar se foi alterado algo que faça com que o programa não funcione da forma correta. Ou seja, sempre que existem atualizações ao programa, estes testes devem ser corridos para verificar se tudo se encontra da forma que é suposto.

Foram realizados testes unitários para todos os métodos que a equipa achou necessário/essencial, sendo estes métodos aqueles que asseguram o correto funcionamento da aplicação. Para a realização dos testes, foi criada uma nova classe só de testes unitários, para que a classe principal não ficasse com métodos que apenas iriam criar confusão.

Os testes unitários encontram-se separados também por métodos, sendo que cada método da aplicação contém um método de teste correspondente.

```java
// Testa o método para saber quantos dias tem um mês num dado ano
private static boolean testHowManyDaysTheMonthHave(int year, int month, int expectedResult) {
    return Covid19Analysis.howManyDaysTheMonthHave(year, month) == expectedResult;
}
```

Figura 22 – Exemplo de um método de teste que tem como função testar um método da aplicação principal

Observando Figura 22, podemos ver que este método recebe os parâmetros necessários a passar para o método a testar e também um parâmetro que contém o resultado esperado. É feita a comparação entre o valor obtido e o valor esperado, sendo que se forem iguais o método devolve *true*, e *false* se não forem iguais.

Para a utilização destes métodos de teste, são criados valores para serem testados, sendo que estes são calculados manualmente de forma a verificar a sua integridade, vendo assim se a aplicação funciona da forma correta.

```java
System.out.println("Year 2000, Month 2 test" +
        (testHowManyDaysTheMonthHave( year: 2000,  month: 2,  expectedResult: 29) ? OK_MESSAGE : NOT_OK_MESSAGE));
System.out.println("Year 1900, Month 2 test" +
        (testHowManyDaysTheMonthHave( year: 1900,  month: 2,  expectedResult: 28) ? OK_MESSAGE : NOT_OK_MESSAGE));
```

Figura 23 – Exemplos da utilização de um método de teste

Como podemos observar na Figura 23, é passado para o método de teste um ano, um mês e a quantidade de dias que é esperado que o método da aplicação devolva (resultado esperado). Ao realizar um teste, é apresentada uma pequena descrição dos valores que estão a ser testados e depois uma mensagem de sucesso ou de insucesso dependendo se o teste passar ou não.

5. Caso de Estudo

Nesta secção realiza-se um estudo para prever nos 7 de primeiros dias de janeiro de 2022, o número total de não infetados, o número total de infetados, o número total de hospitalizados, o número total de internados em Unidades de Cuidados Intensivos (*UCI*) e o número total de óbitos utilizando-se para isso duas matrizes de transição diferentes.

A população inicial a considerar para prever estes dados é a população do último dia 31 de dezembro de 2021.

5.1. Análise e discussão dos resultados

Para o caso de estudo realizado utilizou-se os dados do dia 31 de dezembro de 2021, que se encontram ilustrados na Tabela 2Tabela 2 - Dados do último dia a considerar.

Data	Não infetados	Infetados	Hospitalizados	Internados UCI	Óbitos
31-12-2021	9969087	178712	1024	145	18

Tabela 2 - Dados do último dia a considerar

Os dados reais do dia 1 a 7 de janeiro de 2022 encontram-se na Tabela 3, é muito importante possuir estes dados para comparar com os que serão previstos com as duas matrizes de transição.

Data	Não infetados	Infetados	Hospitalizados	Internados UCI	Óbitos
01-01-2022	9951559	196223	1023	142	21
02-01-2022	9944382	203322	1081	148	14
03-01-2022	9939750	207859	1167	147	10
04-01-2022	9933809	213749	1203	147	15
05-01-2022	9908402	239098	1251	143	14
06-01-2022	9899960	247440	1311	158	25
07-01-2022	9893098	254240	1353	161	17

Tabela 3 - Dados reais do dia 01 a dia 07 de janeiro de 2022

Através da aplicação criada e da matriz de transição A obteve-se os valores apresentados na Tabela 4 para os 7 de primeiros dias de janeiro de 2022.

Data	Não infetados	Infetados	Hospitalizados	Internados UCI	Óbitos
01-01-2022	9969466,0	176554,3	2257,4	684,1	24,1
02-01-2022	9969783,1	174496,2	3461,8	1202,2	42,7
03-01-2022	9970041,2	172533,1	4638,1	1700,2	73,4
04-01-2022	9970243,2	170660,9	5787,1	2179,1	115,8
05-01-2022	9970391,6	168875,4	6909,5	2640,0	169,4
06-01-2022	9970489,2	167172,8	8006,3	3083,7	234,0
07-01-2022	9970538,2	165549,4	9078,0	3511,1	309,2

Tabela 4 - Dados previstos utilizando a matriz de transição A

No caso da matriz de transição B obteve-se os valores apresentados na Tabela 5 para os 7 de primeiros dias de janeiro de 2022.

Data	Não infetados	Infetados	Hospitalizados	Internados UCI	Óbitos
01-01-2022	9972864,6	174768,7	1167,3	163,9	21,6
02-01-2022	9976447,8	171025,5	1304,5	182,6	25,6
03-01-2022	9979846,6	167472,2	1436,0	201,1	30,1
04-01-2022	9983070,2	164099,3	1562,0	219,4	35,1
05-01-2022	9986127,5	160897,7	1682,8	237,5	40,5
06-01-2022	9989027,1	157858,5	1798,7	255,3	46,4
07-01-2022	9991776,9	154973,7	1909,9	272,8	52,6

Tabela 5 - Dados previstos utilizando a matriz de transição B

Numa primeira análise aos dados previstos para os 7 de primeiros dias de janeiro de 2022 utilizando a matriz de transição A percebesse que os dados se tornam bastante irrealistas em particular no número total de hospitalizados, internados UCI e óbitos.

No caso dos dados presumidos com a utilização da matriz de transição B estes estão mais realistas, apesar do número de infetados estar numa queda bastante acentuada.

Para uma melhor comparação entre os valores reais com os previstos, realizou-se o cálculo do erro nos dados. Na Tabela 6 encontra-se o erro nos dados usando a matriz de transição A.

Data	Não infetados	Infetados	Hospitalizados	Internados UCI	Óbitos
01-01-2022	17907	19668,7	1234,4	542,1	3,1
02-01-2022	25401	28825,8	2380,8	1054,2	28,7
03-01-2022	30291	35325,9	3471,1	1553,2	63,4
04-01-2022	36434	43088,1	4584,1	2032,1	100,8
05-01-2022	61990	70222,6	5658,5	2497	155,4
06-01-2022	70529	80267,2	6695,3	2925,7	209
07-01-2022	77440	88690,6	7725	3350,1	292,2

Tabela 6 - Erro nos dados usando a matriz de transição A

Na Tabela 7 encontra-se o erro nos dados usando a matriz de transição B.

Data	Não infetados	Infetados	Hospitalizados	Internados UCI	Óbitos
01-01-2022	21306	21454,3	144,3	21,9	0,6
02-01-2022	32066	32296,5	223,5	34,6	11,6
03-01-2022	40097	40386,8	269	54,1	20,1
04-01-2022	49261	49649,7	359	72,4	20,1
05-01-2022	77726	78200,3	431,8	94,5	26,5
06-01-2022	89067	89581,5	487,7	97,3	21,4
07-01-2022	98679	99266,3	556,9	111,8	35,6

Tabela 7 - Erro nos dados usando a matriz de transição B

Ao compararmos os erros nos dados das 2 previsões conseguimos verificar que os valores que se aproximam mais dos dados reais é quando usamos a matriz de transição B em particular no número total de hospitalizados, internados UCI e óbitos.

Pelo contrário em relação ao número total de não infetados e infetados os dados são mais realistas quando se usa a matriz de transição A.

6. Conclusão

Ao longo do relatório abordou-se diversos tópicos que foram essenciais no decorrer do desenvolvimento do trabalho.

Na metodologia de trabalho abordou-se o *Scrum*, o planeamento e distribuição de tarefas e uma reflexão sobre a dinâmica da equipa. No tópico das cadeias de *Markov* e da decomposição *LU* pelo método de *Crout* explicou-se o seu funcionamento geral e também a sua utilidade. No ponto seguinte clarificou-se a organização dos módulos do projeto, o funcionamento da aplicação no modo interativo e não interativo e uma explicação de algumas implementações importantes na aplicação.

Relativamente ao caso de estudo realizado foi muito interessante perceber que apesar das duas matrizes de transição serem bastantes semelhantes obteve-se valores muito diferentes em relação aos primeiros 7 dias de janeiro de 2022.

Com este trabalho todo o grupo percebeu da melhor forma como funciona o processo básico de desenvolvimento de aplicações informáticas valorizando todas as suas fases desde a análise e conceção aos testes de validação.

Após uma análise ao trabalho desenvolvido existem algumas sugestões de aplicabilidade e melhoria futura como:

- Permitir ao utilizador escolher várias opções na escolha do tipo de dados a visualizar (número de infetados, número de hospitalizados, número de internados em UCI e número de óbitos). Atualmente o utilizador apenas podes escolher um tipo de dados ou todos;
- Possibilitar ao utilizador visualizar qual é o ficheiro que se encontra atualmente carregado em memória para cada um dos possíveis *arrays* com dados.

7. Referências

Markov Chains. (2018). Obtido em 10 de janeiro de 2022, de https://setosa.io/ev/markov-chains/

William H. Press, S. A. (2002). *Numerical Recipes in C.* University of Cambridge.

ANEXOS

ANEXO A

```java
public static final int TOTAL_FIELDS = 8;
public static final int FIELDS_WITHOUT_DATE = 5;
public static final int START_INDEX_WITHOUT_DATE = TOTAL_FIELDS - FIELDS_WITHOUT_DATE;
public static final int YEAR_INDEX = 0;
public static final int MONTH_INDEX = 1;
public static final int DAY_INDEX = 2;
public static final int UNINFECTED_INDEX = 3;
public static final int INFECTED_INDEX = 4;
public static final int HOSPITALIZED_INDEX = 5;
public static final int ICU_ADMITTED_INDEX = 6;
public static final int CASUALTIES_INDEX = 7;
```

Figura 24 – Definição de constantes para aceder às diferentes posições do *array* bidimensional com os dados Covid-19, e úteis para trabalhar o acesso a essas mesmas posições

```java
// Método para saber se o valor de um determinado índice do array é o primeiro dia de um mês e se este mesmo array
// contém os restantes dias desse mesmo mês (se é um mês completo)
public static boolean itsFirstDayAndHaveFullMonth(int[][] dateInterval, int startingDay, int arrayLengthWithData) {
    boolean itsFirstDayAndHaveFullMonth = false;
    if (startingDay >= 0 && startingDay < arrayLengthWithData) {
        int amountOfDaysActualMonth = howManyDaysTheMonthHave(dateInterval[startingDay][YEAR_INDEX], dateInterval[startingDay][MONTH_INDEX]);
        itsFirstDayAndHaveFullMonth = dateInterval[startingDay][DAY_INDEX] == 1
                && (startingDay + amountOfDaysActualMonth - 1 < arrayLengthWithData);
    }
    return itsFirstDayAndHaveFullMonth;
}
```

Figura 25 – Método para verificar se um determinado índice do *array* é o primeiro dia do mês e se existem dados para todos os dias deste mesmo mês

```java
// Método para descobrir qual é o possível índice do primeiro dia do próximo mês
public static int findNextMonthIndex(int[][] dateInterval, int actualIndex) {
    int amountOfDaysActualMonth = howManyDaysTheMonthHave(dateInterval[actualIndex][YEAR_INDEX], dateInterval[actualIndex][MONTH_INDEX]);
    actualIndex += (amountOfDaysActualMonth - dateInterval[actualIndex][DAY_INDEX]) + 1;
    return actualIndex;
}
```

Figura 26 – Método para encontrar o possível índice do próximo mês

```java
// Método para calcular a soma das estatísticas dia a dia
public static int[][] calculateSumOfEveryDayStatistics(int[][] diaryCovidDataInInterval,
                                                       int numberOfLines, String typeSum) {
    int[][] sumOfEveryDayStatistics = new int[numberOfLines][TOTAL_FIELDS];
    int totalIterations = 0, day = 0, intervalLength = 0;

    while (totalIterations < numberOfLines) {
        sumOfEveryDayStatistics[totalIterations][YEAR_INDEX] = diaryCovidDataInInterval[day][YEAR_INDEX];
        sumOfEveryDayStatistics[totalIterations][MONTH_INDEX] = diaryCovidDataInInterval[day][MONTH_INDEX];
        sumOfEveryDayStatistics[totalIterations][DAY_INDEX] = diaryCovidDataInInterval[day][DAY_INDEX];

        if (typeSum.equals(WEEKLY_TYPE)) {
            intervalLength = day + 7;
        } else if (typeSum.equals(MONTHLY_TYPE)) {
            intervalLength = day + howManyDaysTheMonthHave(sumOfEveryDayStatistics[totalIterations][YEAR_INDEX],
                    sumOfEveryDayStatistics[totalIterations][MONTH_INDEX]);
        }

        while (day < intervalLength) {
            for (int data = START_INDEX_WITHOUT_DATE; data < TOTAL_FIELDS; data++) {
                sumOfEveryDayStatistics[totalIterations][data] += diaryCovidDataInInterval[day][data];
            }
            day++;
        }

        totalIterations++;
    }

    return sumOfEveryDayStatistics;
}
```

Figura 27 – Método para calcular a soma das estatísticas semanais ou mensais

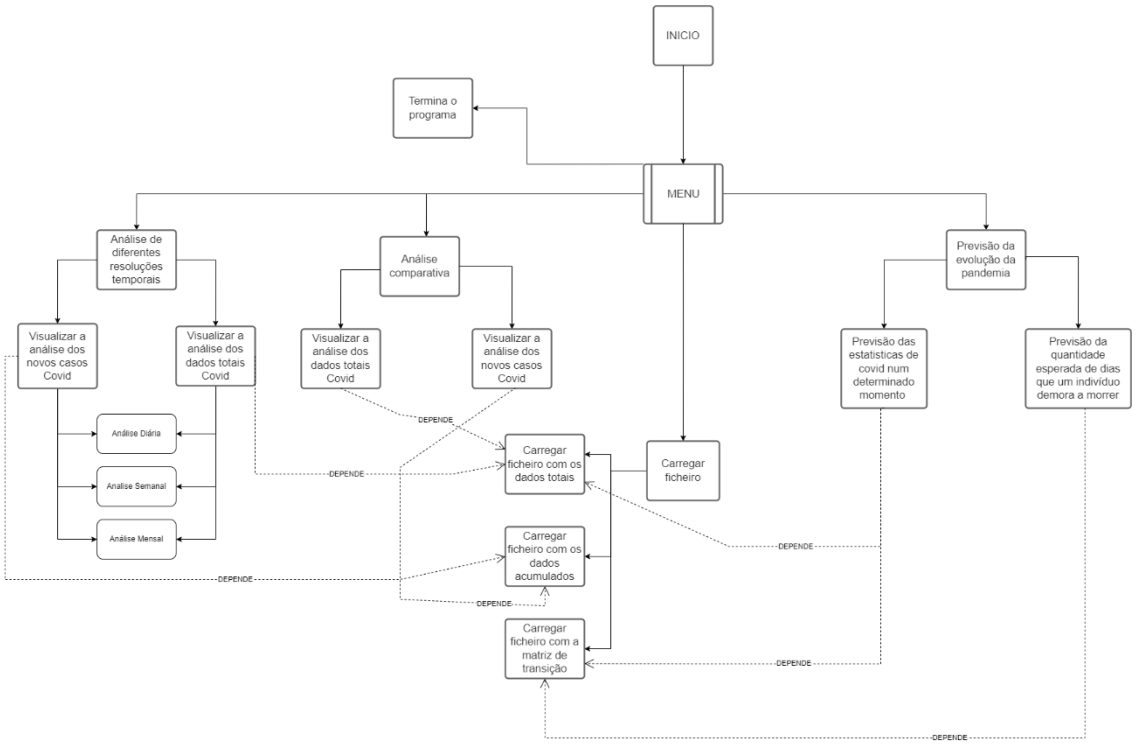

Figura 28 – Diagrama de funcionamento da aplicação