RELATÓRIO DO PROJETO COMPUTACIONAL – Grupo 1

ALEXANDRE CARDEIRA, CÁTIA ANTUNES, FRANCISCO OLIVEIRA, LUÍS ISIDRO, JOSÉ MARTINS

Este relatório descreve as funções utilizadas para responder às questões (A), (B) e (C) do projeto computacional.

CONTENTS

1.Introdução 1

2.Métodos 2

3. Resultados 16

4. Conclusão 21

5. Bibliografia 21

ABSTRACT. This report contains a collection of Octave routines applied to generate and classify magma structures to study some general properties of the cartesian product on those structures.

1. INTRODUÇÃO

Seja A um conjunto qualquer. Uma estrutura de magma sobre o conjunto A é um sistema algébrico (A, m) com $m: A \times A \rightarrow A$ sobre uma função de $A \times A$ em A que se representa por $m(x, y) = x.y$.

Para (A, m), uma estrutura de magma, pode-se analisar as seguintes condições, as quais devem ser válidas para todo $x, y, z, w \in A$:

Comutativa: $x.y = y.x$

Associativa: $x.(y.z) = (x.y).z$

Cancelativa à direita: se existe um $a \in A$ tal que $x.a = y.a$, então $x = y$

Cancelativa à esquerda: se existe um $a \in A$ tal que $a.x = a.y$, então $x = y$

Medial: $(x.y).(z.w) = (x.z).(y.w)$

Elemento Neutro: existe $e \in A$ tal que $e.x = x = x.e$

Date: May 31, 2022; Matemática Discreta EI-PL; Grupo 1

Se (A,m) e (A',m') são duas estruturas de magma, um homomorfismo é uma função $f:(A,m)\to(A',m')$, se e só se a condição

$$f\big(m(x,y)\big)=m'(f(x),f(y)),$$

é verificada para todo o $x,y\in A$.

Se trocarmos os valores de x e y forem trocados no segundo membro da equação acima, podemos definir a noção de anti-isomorfismo através da condição

$$f\big(m(x,y)\big)=m'(f(y),f(x)),$$

Quando f é uma função bijetiva, ou seja, tem uma inversa, e quando satisfaz a condição de homomorfismo, então f diz-se um isomorfismo e os semigrupos (A,m) e (A',m') são denominados de isomorfos. Da mesma forma, quando f é bijetiva e existe um anti-isomorfismo entre (A,m) e (A',m'), então trata-se de um anti-isomorfismo.

Por fim, considera-se que duas estruturas são equivalentes se forem isomorfas ou anti-isomorfas.

2. MÉTODOS

Nesta secção são apresentados todos os métodos utilizados em cada umas das questões do projeto.

Neste trabalho foram considerados os seguintes conjuntos:

$$A=\{1,2\}$$

$$B=\{1,2,3\}$$

tendo sido geradas todas as possíveis estruturas que podem ser definidas nos conjuntos, definindo assim M_2 e M_3. Para efeitos deste projeto, definiu-se uma estrutura em A, como uma matriz 2-por-2 com entradas em 1:2 e, em B, como uma estrutura de 3-por-3 com entradas em 1:3.

2.1 Questões

A função *criarMatriz,* é uma função que permite representar M_2 e M_3, sendo que estas matrizes representam todas as estruturas que podem ser geradas a partir de A e B, respetivamente, cuja cardinalidade é,

$$\#A=2^4=16,$$

$$\#B=3^9=19683.$$

2.1.1 *criarMatriz.m:* Função que recebe um tamanho como paramento e que gera todas as matrizes lineares que podem ser geradas a partir de um dado conjunto.

```
function MLinear = criarMatriz(tamanho)
  #cria matriz em formato linear. ATENÇÃO: função adaptada apenas para os
      casos de 2x2, 3x3
  if(tamanho == 2)
    [a,b,c,d] = ndgrid(1:2,1:2,1:2,1:2);
    MLinear = [a(:), b(:), c(:), d(:)];

    for i=1:16
    %caracteristica homologica
      Mhomologica(i,:) = sort(hist(MLinear(i,:),2));
    end

    [u,v,w]=unique(Mhomologica, 'rows');

    Dados = [MLinear, (1:16)' Mhomologica w];
    MLinear';

    M2x2 = reshape(MLinear',2,2,16);
    % u guarda as classes possiveis (0 4 ; 1 3 ; 2 2)
    % v representa as primeiras vezes que cada classe aparece na matriz
        homologica
    % w guarda a categoria da classe homologica (1,2 ou 3)
    % Mhomologica(1,1) - guarda a 1 posicao da classe da 1a matriz
    % Mhomologica(1,2) - guarda a 2 posicao da classe da 1a matriz
    sortrows(Dados, 8);

    return
  endif

  if(tamanho == 3)
    [a, b, c, d, e, f, g, h, i] = ndgrid(1:3,1:3,1:3,1:3,1:3,1:3,1:3,1:3,1
        :3);
    MLinear = [a(:), b(:), c(:), d(:), e(:), f(:), g(:), h(:), i(:)];
    return
  endif

  if(tamanho == 4)
    [a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p] = ndgrid(1:4,1:4,1:4
        ,1:4,1:4,1:4,1:4,1:4,1:4,1:4,1:2,1,1,1,1,1);

    MLinear = [a(:), b(:), c(:), d(:), e(:), f(:), g(:), h(:), i(:), j(:),
        k(:), l(:), m(:), n(:), o(:), p(:)];
      return
    endif

    disp("So são aceites tamanhos de 2 ou 3")

  end
```

2.1.2 *verificaClassess.m:* Função que recebe como parâmetro uma matriz e que devolve as suas classes, de acordo com a característica homológica, imprimindo o resultado.

```
function verificaClasses(MLinear)
  #dada uma matriz, é verificada a caracteristica homológica de cada matriz
  #e são distinguidas num vetor de contadores de cada classe

  i2s=@(t) MLinear(t,:);

  [a b c] = size(MLinear);
  linhas = sqrt(b); #tamanho das matrizes (2 ou 3)

  for i=1:(size(MLinear,1))
    Mhomologica(i,:)=sort(hist(MLinear(i,:),linhas));
  endfor

  [u,v,w]=unique(Mhomologica,'rows'); # matriz com possibilidades de
      homologia dividido em classes

  tamClasses = size(u,1);

  vetorContaHomologicas = zeros(1,tamClasses);

  for i=1:(size(MLinear,1))
    x=i2s(i); #devolve matriz no indice passado por parametro
    betaX = sort(hist(x,linhas)); #devolve caracteristica homologica da
        matriz x
    z=find(ismember(u,betaX,'rows'));#indice da matriz beta em u
    vetorContaHomologicas(z)++;
  endfor

  vetorContaHomologicas
  disp("Classes: ")
  disp(u)

end
```

Por forma a testar todas as propriedades de cada matriz linear, foram criadas várias funções, em que cada representa a validação de uma condição e aceita como parâmetro uma linha da matriz linear, devolvendo *true* ou *false*. Estas rotinas são de seguida executadas numa função geral que permite avaliar as propriedades de toda a estrutura, devolvendo uma matriz com informação sobre a validação das diferentes propriedades nas diferentes linhas.

2.1.3 *isComutativa.m:* Função que verifica se uma matriz é comutativa.

```
function tf = isComutativa(MatrizLinear)

% recebe uma linha da matriz linear, transformamos para uma matriz quadrada e
    verifica a propriedade

[a b c] = size(MatrizLinear);

linhas = sqrt(b);
Matrix=reshape(MatrizLinear',linhas,linhas);

for x=1:linhas
    for y=1:linhas
        %comutativa
        Left = Matrix(x,y);
        Right = Matrix(y,x);

        if Left ~= Right
            #disp([x, y Left Right]);
            #disp("Matriz nao é comutativa");
            tf = false;
            return
        endif
    end
end
tf = true;
#disp("Matriz é comutativa")
#Matrix
```

2.1.4 *isAssociativa.m:* Função que avalia se uma matriz verifica a associatividade.

```
function tf = isAssociativa(MatrizLinear)

% recebe uma linha da matriz linear, transformamos para uma matriz quadrada e
    verifica a propriedade

[a b c] = size(MatrizLinear);

linhas = sqrt(b);
Matrix=reshape(MatrizLinear',linhas,linhas);

for x=1:linhas
    for y=1:linhas
        for z=1:linhas
            %associativa
            Left=Matrix(x,Matrix(y,z));
            Right=Matrix(Matrix(x,y),z);

            if Left ~= Right
                #disp([x, y, z, Left Right]);
                #disp("Matriz não é associativa")
                tf = false;
                return
            endif
        end
    end
end
tf = true;
#disp("Matriz é associativa");
Matrix;
```

2.1.5 *isMedial.m:* Função que verifica se uma matriz é medial.

```
function tf = isMedial(MatrizLinear)

% recebe uma linha da matriz linear, transformamos para uma matriz quadrada e
    verifica a propriedade

[a b c] = size(MatrizLinear);

linhas = sqrt(b);
Matrix=reshape(MatrizLinear',linhas,linhas);

for x=1:linhas
    for y=1:linhas
        for z=1:linhas
            for w=1:linhas
                %medial
                Left = Matrix(Matrix(x,y),Matrix(z,w));
                Right = Matrix(Matrix(x,z),Matrix(y,w));

                if Left ~= Right
                    #disp([x, y, z, w Left Right]);
                    #disp("Matriz não é medial")
                    tf = false;
                    return
                endif
            end
        end
    end
end
tf = true;
#disp("Matriz é medial")
Matrix;
```

2.2.6 *hasElementoNeutro.m*: Função que recebe como parâmetro uma matriz e verifica se esta possui um elemento neutro.

```
function tf = hasElementoNeutro(MatrizLinear)

% recebe uma matriz linear, transforma em matrix quadrada e verifica a
    propriedade

[a b c] = size(MatrizLinear);

linhas = sqrt(b);
Matrix=reshape(MatrizLinear',linhas,linhas);

for e=1:linhas
    count = 0;
    for x=1:linhas
          Left = Matrix(e,x);
          Right = Matrix(x,e);

          if Left == x
            if Right == x
              count++;
            endif
          endif

          if count == linhas
             #disp("El neutro: ");
             #disp(e);
             tf = true;
             return
          endif

    end
end
tf = false;

#disp("Esta matriz não tem elemento neutro.");

end
```

2.1.7 *isCancelativaEsquerda.m*: Função que recebe como parâmetro uma linha da matriz linear (matriz 2x2 ou 3x3 linearizada) e compara com um vetor pré-definido de acordo com a propriedade, avaliando se são iguais.

```
function tf = isCancelativaEsquerda(MatrizLinear)

% recebe uma linha da matriz linear, transformamos para uma matriz quadrada
    verifica a propriedade

[a b c] = size(MatrizLinear);

linhas = sqrt(b);
MatrizLinear;
Matrix=reshape(MatrizLinear',linhas,linhas);

if(linhas == 2)
    vetor = [1 2];
endif

if(linhas == 3)
    vetor = [1 2 3];
endif


for x=1:linhas
    for y=1:linhas
    %cancelativaEsquerda
        mfinal(x,y) = Matrix(x,y);
    end
    mfinal(x,:);
    vetor;
    a = sort(mfinal(x,:),2);
    a;
    if(!isequal(a, vetor))
        #disp("Não é cancelativa à esquerda")
        tf = false;
        return
    endif
end
tf = true;
#disp("É cancelativa à esquerda")
Matrix;
```

2.1.8 isCancelativaDireita.m: Função que recebe como parâmetro uma linha da matriz linear (matriz 2x2 ou 3x3 linearizada) e compara com um vetor pré-definido de acordo com a propriedade, avaliando se são iguais.

```
function tf = isCancelativaDireita(MatrizLinear)

% recebe uma linha da matriz linear, transformamos para uma matriz quadrada e
    verifica a propriedade


[a b c] = size(MatrizLinear);

linhas = sqrt(b);
MatrizLinear;
Matrix=reshape(MatrizLinear',linhas,linhas);

if(linhas == 2)
    vetor = [1 2];
endif

if(linhas == 3)
    vetor = [1 2 3];
endif


for y=1:linhas
    for x=1:linhas
    %cancelativaDireita
        mfinal(x,y) = Matrix(x,y);
    end
    a = sort(mfinal(:,y));
    a';
    vetor;
    if(!isequal(a', vetor))
        #disp("Não é cancelativa à direita")
        tf = false;
        return
    endif
end
tf = true;
#disp("É cancelativa à direita")
Matrix;
```

2.1.9 *procCancEsquerda.m:* Função que recebe uma matriz, transformando-a num vetor de n por n de 3 dimensões, comparado se é igual ao vetor predefinido, de acordo com o número de linhas. Devolve o índice da matriz e o total de matrizes que verificam a propriedade.

```
function procCancEsquerda(MatrizLinear)

% Recebe uma matriz linear, copia as suas linhas para um vetor n por n de 3
    dimensões e verifica quais são cancelativas à esquerda devolvendo o indice da
    matriz

[a b c] = size(MatrizLinear);

linhas = sqrt(b);

if (linhas == 3)
    vetor = [1 2 3; 1 2 3; 1 2 3];
endif

if (linhas == 2)
    vetor = [1 2; 1 2];
endif

cont=0;
for i=1:a
    M3d(:,:,i)=reshape(MatrizLinear(i,:)',linhas,linhas);

    %cancelativa à esquerda
    a = sort(M3d(:,:,i),2);

    if(isequal(a, vetor))
        a;
        M3d(:,:,i);
        disp("indice da matriz")
        i
        cont++;
    endif
endfor

cont
```

2.1.10 *procCancDireita.m*: Função que recebe uma matriz, transforma essa estrutura num vetor de n por n de 3 dimensões, comparado de seguida se é igual ao vetor predefinido, de acordo com o número de linhas. Devolve o índice da matriz e o total de matrizes que verificam a propriedade.

```
function procCancDireita(MatrizLinear)

% Recebe uma matriz linear, copia as suas linhas para um vetor n por n de 3
    dimensões e verifica quais são cancelativas à direita devolvendo o indice
    da matriz

[a b c] = size(MatrizLinear);

linhas = sqrt(b);

if (linhas == 3)
    vetor = [1 2 3; 1 2 3; 1 2 3];
endif

if (linhas == 2)
    vetor = [1 2; 1 2];
endif


cont=0;
for i=1:a
    M3d(:,:,i)=reshape(MatrizLinear(i,:),linhas,linhas);

    %cancelativa à direita
    a = sort(M3d(:,:,i));

    if(isequal(a, vetor'))
        a;
        M3d(:,:,i);
        disp("indice da matriz")
        i
        cont++;
    endif
endfor

cont
```

2.1.11 *verificaPropriedades.m*: Função que recebe como parâmetro uma matriz linearizada e que tem como objetivo verificar as propriedades em todas as suas linhas, devolvendo uma estrutura que permitir observar quais é que verificam as propriedades ou não.

```
function verificaPropriedades(matriz)

% recebe a matriz linearizada, e devolve os seus valores nas primeiras "b"
    casas, a propriedade isAssociativa, isComutativa, isMedial,
    isCancelativaEsquerda, isCancelativaDireita e, hasElementoNeutro
    respetivamente.

[a b c] = size(matriz);

vetorAssociativa = zeros(1,a);
vetorComutativa = zeros(1,a);
vetorMedial = zeros(1,a);
vetorCancEsquerda = zeros(1,a);
vetorCancDireita = zeros(1,a);
vetorElemNeutro = zeros(1,a);

for i=1:a
    a = isAssociativa(matriz(i,:));
    b = isComutativa(matriz(i,:));
    c = isMedial(matriz(i,:));
    d = isCancelativaEsquerda(matriz(i,:));
    e = isCancelativaDireita(matriz(i,:));
    f = hasElementoNeutro(matriz(i,:));


    vetorAssociativa(i) = a;
    vetorComutativa(i) = b;
    vetorMedial(i) = c;
    vetorCancEsquerda(i) = d;
    vetorCancDireita(i) = e;
    vetorElemNeutro(i) = f;
endfor

[matriz, vetorAssociativa' vetorComutativa' vetorMedial' vetorCancEsquerda'
    vetorCancDireita' vetorElemNeutro']
```

2.1.12 *isIso.m:* Função que recebe como parâmetros uma matriz linearizada com todas as matrizes possíveis (MLinear), um vetor com uma permutação possível (alpha) e um índice que aponta para uma linha da nossa matriz. Esta rotina calcula a matriz a ser usada pela procura na matriz MLinear através do índice, usando-a então para calcular a matriz que lhe é isomorfa. Posteriormente, é verificado se esta matriz existe dentro da nossa matriz MLinear e é-nos devolvido o seu índice.

```
function isIso(MLinear,alpha,indMLinear)
   #recebe uma matriz NxN linearizada, um alpha e um índice de uma matriz na
       matriz principal criada

    tmp = size(alpha);
    tam = tmp(2);           #tamanho das matrizes

    i2s=@(x) MLinear(x,:);
    aLinear=i2s(indMLinear);
    aMatrix=reshape(aLinear',tam,tam);
    aIsoMatrix = zeros(tam);
    alphainv(alpha)=1:tam;

    for k=1:tam
      for l=1:tam
        aIsoMatrix(k,l)=alphainv(aMatrix(alpha(k),alpha(l)));
      endfor
    endfor


    aIsoLinear = reshape(aIsoMatrix,1,[]);
    indice=find(ismember(MLinear,aIsoLinear,'rows'))
    i2s(indice)

end
```

2.1.13 *isIso2.m*: Função semelhante à anterior, no entanto recebe como parâmetro apenas a matriz linearizada com todas as matrizes possíveis (MLinear). Neste caso calculamos todas as possibilidades de permutação dentro da própria matriz, devolvendo uma listagem, em coluna, das diferentes classes de isomorfismo.

```
function DadosIso = isIso2(MLinear)
   #função semelhante à isIso, no entanto esta calcula logo todas as
       possibilidades de permutação
   #dentro da própria função e devolve uma listagem em coluna das
       diferentes classes de isomorfismo


    [a b] = size(MLinear);

    tam = sqrt(b);
                #tamanho das matrizes

    Perms = sortrows(perms(1:tam));
    [e] = size(Perms);
    o = e(1);

    i2s=@(x) MLinear(x,:);

    m = zeros(1,a)';

    Dados = [MLinear, m];
    class = 1;

    for p=1:a

      aLinear=i2s(p);
      aMatrix=reshape(aLinear',tam,tam);
      aIsoMatrix = zeros(tam);
      contHelper = 0;
      for u=1:o
        alpha = Perms(u,:);
        alphainv(alpha)=1:tam;
        for k=1:tam
          for l=1:tam
            aIsoMatrix(k,l)=alphainv(aMatrix(alpha(k),alpha(l)));
          endfor
        endfor
        aIsoLinear = reshape(aIsoMatrix,1,[]);
        indice=find(ismember(MLinear,aIsoLinear,'rows'));
        i2s(indice);
        if(Dados(indice,b+1) == 0)
          Dados(indice,b+1) = class;
          continue;

        endif
        contHelper++;
      endfor
      if contHelper != 2
        class++;
      endif


    endfor
    Dados;
    DadosIso = Dados(:,b+1);

end
```

2.1.14 *isAntiIso.m:* Função que recebe como parâmetros uma matriz linearizada com todas as matrizes possíveis (MLinear), um vetor com uma permutação possível (alpha) e um índice que aponta para uma linha da nossa matriz. Esta rotina calcula a matriz a ser usada pela procura na matriz MLinear através do índice, usando-a então para calcular a matriz que lhe é isomorfa como também a que lhe é anti-isomorfa. Posteriormente, é verificado se estas matrizes existem dentro da nossa matriz MLinear e é-nos devolvido os seus índices.

```
function isAntiIso(MLinear,alpha,indMLinear)
   #recebe uma matriz NxN linearizada, um alpha e um índice de uma matriz na
       matriz principal criada
   #verifica iso e antiiso

    tmp = size(alpha);
    tam = tmp(2); #tamanho das matrizes

    i2s=@(x) MLinear(x,:);
    aLinear=i2s(indMLinear);
    aMatrix=reshape(aLinear',tam,tam);
    aAntiIsoMatrix = zeros(tam);
    aIsoMatrix = zeros(tam);
    alphainv(alpha)=1:tam;

    for k=1:tam
      for l=1:tam
        aAntiIsoMatrix(k,l)=alphainv(aMatrix(alpha(l),alpha(k)));
        aIsoMatrix(k,l)=alphainv(aMatrix(alpha(k),alpha(l)));
      endfor
    endfor


  aAntiIsoLinear = reshape(aAntiIsoMatrix,1,[]);
  aIsoLinear = reshape(aIsoMatrix,1,[]);



  disp("Matriz anti-isomorfica")
  indice1=find(ismember(MLinear,aAntiIsoLinear,'rows'))
  i2s(indice1) #matriz anti-isomorfica
  disp("Matriz isomorfica")
  indice2=find(ismember(MLinear,aIsoLinear,'rows'))
  i2s(indice2) #matriz isomorfica

end
```

2.1.15 *isAntiIso2.m:* Função semelhante à anterior, no entanto recebe como parâmetro apenas a matriz linearizada com todas as matrizes possíveis (MLinear). Neste caso calculamos todas as possibilidades de permutação dentro da própria matriz, devolvendo uma listagem, em coluna, das diferentes classes resultantes da combinação de isomorfismo com anti-isomorfismo.

```
function DadosAntiIso = isAntiIso2(MLinear)
   #função semelhante à isAntiIso, no entanto esta calcula logo todas as
       possibilidades de permutação
   #dentro da própria função e devolve uma listagem em coluna das
       diferentes classes de isomorfismo
   # e antiisomorfismo

    [a b] = size(MLinear);

    tam = sqrt(b);
                  #tamanho das matrizes

    Perms = sortrows(perms(1:tam));
    [e] = size(Perms);
    o = e(1);

    i2s=@(x) MLinear(x,:);

    m = zeros(1,a)';

    Dados = [MLinear, m];
    class = 1;

    for p=1:a

      aLinear=i2s(p);
      aMatrix=reshape(aLinear',tam,tam);
      aIsoMatrix = zeros(tam);
      aAntiIsoMatrix = zeros(tam);
      contHelper = 0;
      for u=1:o
        alpha = Perms(u,:);
        alphainv(alpha)=1:tam;
        for k=1:tam
          for l=1:tam
            aAntiIsoMatrix(k,l)=alphainv(aMatrix(alpha(l),alpha(k)));
            aIsoMatrix(k,l)=alphainv(aMatrix(alpha(k),alpha(l)));
          endfor
        endfor
        aIsoLinear = reshape(aIsoMatrix,1,[]);
        aAntiIsoLinear = reshape(aAntiIsoMatrix,1,[]);
        indice1=find(ismember(MLinear,aIsoLinear,'rows'));
        indice2=find(ismember(MLinear,aAntiIsoLinear,'rows'))
```

```
        if(Dados(indice1,b+1) == 0)
          Dados(indice1,b+1) = class;
        end

        if(Dados(indice2,b+1) == 0)
          Dados(indice2,b+1) = class;
        end


        #contHelper++;
      endfor
      tmp = max(Dados(:,b+1));
      #if contHelper != 2
      #  class++;
      #endif
      if tmp == class
        class++;
      endif

    endfor
    Dados;
    DadosAntiIso = Dados(:,b+1);
end
```

2.1.16 *adicionaPropriedades.m:* Esta função tem o objetivo de, dada a matriz linearizada com todas as matrizes possíveis (MLinear), permitir-nos calcular as diferentes classes identificativas das relações de equivalência de homologia, isomorfismo e isomorfismo com anti-isomorfismo. Após o cálculo, é apresentado no ecrã a matriz linear com as várias colunas à direita representativas das classes que foram calculadas.

```
function MPropriedades = adicionaPropriedades(MLinear)
  #função que dadas todas as matrizes linearizadas, adiciona colunas à sua
    direita
 #associadas às classes definidas pela relação de equivalência por
    homologia, isomorfismo
 #e antiIsomorfismo ou Isomorfismo

  tmp60 = size(MLinear);
  numMatrizes = tmp60(1);
  tam = sqrt(tmp60(2));
  for i=1:numMatrizes
    %caracteristica homologica
      Mhomologica(i,:) = sort(hist(MLinear(i,:),tam));
    end

    [u,v,w]=unique(Mhomologica, 'rows');

    Dados = [MLinear, (1:numMatrizes)' Mhomologica w];
    MLinear';

    M2x2 = reshape(MLinear',tam,tam,numMatrizes);
    % u guarda as classes possiveis (0 4 ; 1 3 ; 2 2)
    % v representa as primeiras vezes que cada classe aparece na matriz
        homologica
    % w guarda a categoria da classe homologica (1,2 ou 3)

    % Mhomologica(1,1) - guarda a 1 posicao da classe da 1a matriz
    % Mhomologica(1,2) - guarda a 2 posicao da classe da 1a matriz
    %MLinear = sortrows(Dados, 8);
    #function isIso2


    tmp=size(u);
    colIso = isIso2(MLinear);
    colAntiIso = isAntiIso2(MLinear);

    MPropriedades = [Dados colIso colAntiIso];

    nClassesIsomorfismo = max(colIso)
    nClassesAntiIsomorfismo = max(colAntiIso)
    nClassesHomologia = max(tmp(1))

end
```

2.1.17 *imprimeIsoAnti.m:* Esta função tem também como objetivo de mostrar as relações de equivalência ao nível do isomorfismo e anti-isomorfismo (como as combinações) de uma forma matricial, que permite ser mais fácil visualização para o utilizador. De notar que esta função está apenas preparada para as matrizes com tamanho 2x2, pelo que, para as matrizes 3x3 achámos que esta função se iria tornar inviável, tanto pela ineficiência do cálculo, como a difícil visualização da matriz gerada.

```
function imprimeIsoAnti(MatrizLin)
  #gera matrizes de relação isomórifica, anti-isomórfica e a sua reunião
     (iso + anti-iso)
  #função apenas funciona para o caso de matrizes 2x2

  Matriz = reshape(MatrizLin',2,2,16);

  alpha=[2 1];
  alphainv(alpha)=1:2;

  Iso = zeros(16);
  Anti = zeros(16);

  for a = 1:16
    for b = 1:16
      A = Matriz(:,:,a);
      B = Matriz(:,:,b);
      countIso = 0;
      countAnti = 0;
      for i=1:2
        for j=1:2
          if( alpha(A(i,j)) == B(alpha(j),alpha(i)) )
            countAnti++;
          endif
          if( alpha(A(i,j)) == B(alpha(i),alpha(j)) )
            countIso++;
          endif
        end
      end

      if(countIso == 4) #verifica isomorfismo
        Iso(a,b) = 1;
      endif

      if(countAnti == 4) #verifica anti-isomorfismo
        Anti(a,b) = 1;
      endif
    end
  end
  Iso #mostra tabela de isomorfismos
  Anti #mostra tabela de antimorfismo
  AntiIso = Iso + Anti; #gera tabela de antimorfismo reunido com iso

    for a = 1:16
      for b = 1:16
        if(AntiIso(a,b) == 2)
          AntiIso(a,b) = 1;
        endif
      endfor
    endfor
    AntiIso
    ################################################
  end
```

2.2.18 *main.c:* É um ficheiro com o procedimento utilizado pelo grupo para executar as rotinas acima mencionadas.

```
tam = 3;                    #define tamanho de matrizes a serem criadas (apenas estão preparados os casos 2 ou 3)
MLinear = criarMatriz(tam); #cria matriz 2x2 ou 3x3
verificaClasses(MLinear)   #demora algum tempo a calcular para as matrizes 3x3 (~2min)
isComutativa(MLinear(1,:));   #verificamos as propriedades comutativa, associativa,
isAssociativa(MLinear(1,:));  #medial, cancelativa à esquerda e cancelativa à direita para uma matriz linear
isMedial(MLinear(1,:));       #passada por parâmetro. (Neste exemplo usámos a 1a matriz linear existente em MLinear)
isCancelativaEsquerda(MLinear(1,:));
isCancelativaDireita(MLinear(1,:));
hasElementoNeutro(MLinear(1,:));
MPerms = perms(1:tam); #perms gera matriz com possibilidades de transformação das matrizes
alpha = MPerms(2,:);   #cada linha de MPerms é uma permutação possível (alpha), para os testes abaixo
indice = 10;           #escolhemos a linha 2 de MPerms como permutação e o índice da matriz 10 em MLinear
isIso(MLinear,alpha,indice);     #verifica se existe alguma matriz isomórfica aquela que é passada na função
isAntiIso(MLinear,alpha,indice); #verifica se existe alguma matriz isomórfica ou antiisomórfica
                                 #aquela que é passada na função

verificaPropriedades(MLinear); # recebe a matriz linearizada, e devolve os seus valores nas primeiras
                               #b casas, a propriedade isAssociativa, isComutativa, isMedial, isCancelativaEsquerda,
                               #isCancelativaDireita, por esta ordem

adicionaPropriedades(MLinear); #função que dadas todas as matrizes linearizadas, adiciona colunas à sua direita
                               #associadas às classes definidas pela relação de equivalência por homologia, isomorfismo
                               #e antiIsomorfismo ou Isomorfismo

imprimeIsoAnti(MLinear); #ATENÇÃO: esta função apenas pode ser chamada para as matrizes 2x2
```

3. RESULTADOS

Nesta secção apresentamos os resultados obtidos para cada uma das funções do projeto computacional.

3.1.1 *criarMatriz.m*

Nas matrizes 2x2, também devolve uma estrutura que na qual as quatro primeiras posições representam uma matriz, a quinta posição é o índice que a permite identificar na estrutura, a sexta e sétima representam a frequência dos elementos na matriz e na oitava posição está a classe homológica a que cada pertence.

```
octave:15> criarMatriz(2)
ans =

    1    1    1    1    1    0    4    1
    2    2    2    2   16    0    4    1
    2    1    1    1    2    1    3    2
    1    2    1    1    3    1    3    2
    1    1    2    1    5    1    3    2
    2    2    2    1    8    1    3    2
    1    1    1    2    9    1    3    2
    2    2    1    2   12    1    3    2
    2    1    2    2   14    1    3    2
    1    2    2    2   15    1    3    2
    2    2    1    1    4    2    2    3
    2    1    2    1    6    2    2    3
    1    2    2    1    7    2    2    3
    2    1    1    2   10    2    2    3
    1    2    1    2   11    2    2    3
    1    1    2    2   13    2    2    3

ans =

   1   1   1   1
   2   1   1   1
   1   2   1   1
   2   2   1   1
   1   1   2   1
   2   1   2   1
   1   2   2   1
   2   2   2   1
   1   1   1   2
   2   1   1   2
   1   2   1   2
   2   2   1   2
   1   1   2   2
   2   1   2   2
   1   2   2   2
   2   2   2   2
```

```
octave:20> criarMatriz(3)
ans =

   1   1   1   1   1   1   1   1   1
   2   1   1   1   1   1   1   1   1
   3   1   1   1   1   1   1   1   1
   1   2   1   1   1   1   1   1   1
   2   2   1   1   1   1   1   1   1
   3   2   1   1   1   1   1   1   1
   1   3   1   1   1   1   1   1   1
   2   3   1   1   1   1   1   1   1
   3   3   1   1   1   1   1   1   1
   1   1   2   1   1   1   1   1   1
   2   1   2   1   1   1   1   1   1
   3   1   2   1   1   1   1   1   1
   1   2   2   1   1   1   1   1   1
   2   2   2   1   1   1   1   1   1
   3   2   2   1   1   1   1   1   1
   1   3   2   1   1   1   1   1   1
   2   3   2   1   1   1   1   1   1
   3   3   2   1   1   1   1   1   1
   1   1   3   1   1   1   1   1   1
   2   1   3   1   1   1   1   1   1
   3   1   3   1   1   1   1   1   1
   1   2   3   1   1   1   1   1   1
   2   2   3   1   1   1   1   1   1
   3   2   3   1   1   1   1   1   1
   1   3   3   1   1   1   1   1   1
   2   3   3   1   1   1   1   1   1
   3   3   3   1   1   1   1   1   1
   1   1   1   2   1   1   1   1   1
   2   1   1   2   1   1   1   1   1
```

3.1.2 *vetorClassess.m*

```
>> verificaClasses(m);
vetorContaHomologicas =

   2   8   6

Classes:
   0   4
   1   3
   2   2
```

```
vetorContaHomologicas =

      3     54    216    504    756    216   1512   3024   1890   2268   7560   1680

Classes:
    0    0    9
    0    1    8
    0    2    7
    0    3    6
    0    4    5
    1    1    7
    1    2    6
    1    3    5
    1    4    4
    2    2    5
    2    3    4
    3    3    3
```

3.1.3 *verificaPropridades.m*

Devolve uma estrutura, que no caso das matrizes *2x2*, as quatro primeiras posições representam uma matriz e as seguintes posições representam a presença das propriedades associativa, comutativa, medial, cancelamento à esquerda, cancelamento à direita e se tem elemento neutro, respetivamente. Nas matrizes *3x3*, as nove primeiras representam a matriz, e as seguintes posições são iguais à matriz *2x2*.

```
octave:3> verificaPropriedades(m2)
ans =

   1   1   1   1   1   1   1   0   0   0
   2   1   1   1   0   1   0   0   0   0
   1   2   1   1   0   0   0   0   0   0
   2   2   1   1   0   0   1   1   0   0
   1   1   2   1   0   0   0   0   0   0
   2   1   2   1   0   0   1   0   1   0
   1   2   2   1   1   1   1   1   1   1
   2   2   2   1   0   1   0   0   0   0
   1   1   1   2   1   1   1   0   0   1
   2   1   1   2   1   1   1   1   1   1
   1   2   1   2   1   0   1   0   1   0
   2   2   1   2   0   0   0   0   0   0
   1   1   2   2   1   0   1   1   0   0
   2   1   2   2   0   0   0   0   0   0
   1   2   2   2   1   1   1   0   0   1
   2   2   2   2   1   1   1   0   0   0
```

```
octave:5> verificaPropriedades(m3)
ans =

   1   1   1   1   1   1   1   1   1   1   1   1   0   0   0
   2   1   1   1   1   1   1   1   1   0   1   0   0   0   0
   3   1   1   1   1   1   1   1   1   0   1   0   0   0   0
   1   2   1   1   1   1   1   1   1   0   0   0   0   0   0
   2   2   1   1   1   1   1   1   1   0   0   0   0   0   0
   3   2   1   1   1   1   1   1   1   0   0   0   0   0   0
   1   3   1   1   1   1   1   1   1   0   0   1   0   0   0
   2   3   1   1   1   1   1   1   1   0   0   0   0   0   0
   3   3   1   1   1   1   1   1   1   0   0   0   0   0   0
   1   1   2   1   1   1   1   1   1   0   0   1   0   0   0
   2   1   2   1   1   1   1   1   1   0   0   0   0   0   0
   3   1   2   1   1   1   1   1   1   0   0   0   0   0   0
   1   2   2   1   1   1   1   1   1   0   0   0   0   0   0
   2   2   2   1   1   1   1   1   1   0   0   1   0   0   0
   3   2   2   1   1   1   1   1   1   0   0   0   0   0   0
   1   3   2   1   1   1   1   1   1   0   0   0   0   0   0
   2   3   2   1   1   1   1   1   1   0   0   0   0   0   0
```

3.1.4 *procCancEsquerda.m*

```
octave:58> procCancEsquerda(M(4,:))
indice da matriz
i = 1
cont = 1
octave:59> procCancEsquerda(M(1,:))
cont = 0
```

```
octave:48> procCancEsquerda([1 3 2 2 1 3 3 2 1])
M3d =

   1   2   3
   3   1   2
   2   3   1

indice da matriz
i = 1
cont = 1
octave:49> procCancEsquerda([1 2 3 1 2 3 1 2 3])
M3d =

   1   1   1
   2   2   2
   3   3   3

cont = 0
```

3.1.5 *procCancDireita.m*

```
octave:15> procCancDireita(M2)
indice da matriz
i = 6
indice da matriz
i = 7
indice da matriz
i = 10
indice da matriz
i = 11
cont = 4
```

```
indice da matriz
i = 15888
indice da matriz
i = 15892
indice da matriz
i = 15896
indice da matriz
i = 15898
cont = 216
```

3.1.6 *isIso.m*

```
>> m2 = criarMatriz(2);
>> isIso(m2,[2 1], 2)
ans =

   2   2   2   1

ans = 8
```

```
>> m3 = criarMatriz(3);
>> isIso(m3,[2 1 3], 5000)
ans =

   1   3   3   1   1   2   3   2   2

ans = 10474
```

3.1.7 *isAntiIso.m*

```
>> m2 = criarMatriz(2);
>> isAntiIso(m2,[2 1], 4)
Matriz anti-isomorfica
ans =

   2   1   2   1

Matriz isomorfica
ans =

   2   2   1   1
```

```
>> m3 = criarMatriz(3);
>> isAntiIso(m3,[2 1 3], 763)
Matriz anti-isomorfica
ans =

   2   3   2   1   2   1   2   2   2

Matriz isomorfica
ans =

   2   1   2   3   2   2   2   1   2
```

3.1.8 *adicionaPropriedades.m*

Devolve uma estrutura, que no caso das matrizes *2x2*, as quatro primeiras posições representam uma matriz, e nas *3x3* são as primeiras nove. No caso, das 2x2, a quinta posição é o índice que a permite identificar na estrutura, a sexta e sétima representam a frequência dos elementos na matriz, na oitava posição está a classe homológica a que cada pertence, a penúltima coluna é a classe resultante do isomorfismo e última coluna representa as classes por anti-isomorfismo e isomorfismo

1	1	1	1	1	0	4	1	1	1
2	1	1	1	2	1	3	2	2	2
1	2	1	1	3	1	3	2	3	3
2	2	1	1	4	2	2	3	4	4
1	1	2	1	5	1	3	2	5	3
2	1	2	1	6	2	2	3	6	4
1	2	2	1	7	2	2	3	7	5
2	2	2	1	8	1	3	2	2	2
1	1	1	2	9	1	3	2	8	6
2	1	1	2	10	2	2	3	7	5
1	2	1	2	11	2	2	3	9	7
2	2	1	2	12	1	3	2	3	3
1	1	2	2	13	2	2	3	10	7
2	1	2	2	14	1	3	2	5	3
1	2	2	2	15	1	3	2	8	6
2	2	2	2	16	0	4	1	1	1

1	3	1	3	3	3	3	3	3	19663	0	2	7	3	109	59
2	3	1	3	3	3	3	3	3	19664	1	1	7	6	190	131
3	3	1	3	3	3	3	3	3	19665	0	1	8	2	28	3
1	1	2	3	3	3	3	3	3	19666	1	2	6	7	379	333
2	1	2	3	3	3	3	3	3	19667	1	2	6	7	460	412
3	1	2	3	3	3	3	3	3	19668	1	1	7	6	298	253
1	2	2	3	3	3	3	3	3	19669	1	2	6	7	622	543
2	2	2	3	3	3	3	3	3	19670	0	3	6	4	365	319
3	2	2	3	3	3	3	3	3	19671	0	2	7	3	541	208
1	3	2	3	3	3	3	3	3	19672	1	1	7	6	136	62
2	3	2	3	3	3	3	3	3	19673	0	2	7	3	217	134
3	3	2	3	3	3	3	3	3	19674	0	1	8	2	55	6
1	1	3	3	3	3	3	3	3	19675	0	2	7	3	325	279
2	1	3	3	3	3	3	3	3	19676	1	1	7	6	406	359
3	1	3	3	3	3	3	3	3	19677	0	1	8	2	244	199
1	2	3	3	3	3	3	3	3	19678	1	1	7	6	568	490
2	2	3	3	3	3	3	3	3	19679	0	2	7	3	649	569
3	2	3	3	3	3	3	3	3	19680	0	1	8	2	487	199
1	3	3	3	3	3	3	3	3	19681	0	1	8	2	82	56
2	3	3	3	3	3	3	3	3	19682	0	1	8	2	163	128
3	3	3	3	3	3	3	3	3	19683	0	0	9	1	1	1

3.1.9 *adicionaPropriedades.m*

Além de devolver uma estrutura igual à da função acima mencionada, também devolve a contagem de classes de acordo com o isomorfismo, anti-isomorfismo e homologia. Os valores são mostrados abaixo para as matrizes 2x2 e 3x3 respetivamente. Os valores dizem respeito à relação de equivalência por homologia, isomorfismo e isomorfismo com anti-isomorfismo respetivamente.

```
3    10    7
```

```
12    19306    1734
```

3.1.10 *imprimeAntiIso.m*

```
>> imprimeIsoAnti(m2)
Iso =

    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    1
    0    0    0    0    0    0    0    1    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    1    0    0    0    0
    0    0    0    1    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    1    0    0
    0    0    0    0    0    1    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    1    0    0    0    0    0    0
    0    1    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    1    0
    0    0    0    0    0    0    1    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    1    0    0    0    0    0
    0    0    1    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    1    0    0    0
    0    0    0    0    1    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    1    0    0    0    0    0    0    0
    1    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0

Anti =

    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    1
    0    0    0    0    0    0    0    1    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    1    0    0
    0    0    0    0    0    1    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    1    0    0    0    0
    0    0    0    1    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    1    0    0    0    0    0    0
    0    1    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    1    0
    0    0    0    0    0    0    1    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    1    0    0    0
    0    0    0    0    1    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    1    0    0    0    0    0
    0    0    1    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    1    0    0    0    0    0    0    0
    1    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0

AntiIso =

    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    1
    0    0    0    0    0    0    0    1    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    1    0    1    0    0
    0    0    0    1    0    1    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    1    0    1    0    0
    0    0    0    1    0    1    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    1    0    0    0    0    0    0
    0    1    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    1    0
    0    0    0    0    0    0    1    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    1    0    1    0    0    0
    0    0    1    0    1    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    1    0    1    0    0    0
    0    0    1    0    1    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    1    0    0    0    0    0    0    0
```

4.CONCLUSÃO

Este projeto computacional foi desenvolvido com o objetivo de explorar as potencialidades da programação do *Octave* no que diz respeito à teoria sobre as estruturas de magma e as suas propriedades, bem como, a classificação das estruturas de semigrupos e a existência de isomorfismo e anti-isomorfismo.

Uma das limitações do projeto é o facto de as funções implementadas só aceitarem estruturas de *2x2* ou *3x3*. A rotina *CriarMatriz.m* apresenta o código para implementar *4x4*, no entanto acorre um transbordo de memória. Além disso, são apresentadas algumas rotinas que apenas podem ser utilizadas nas estruturas *2x2*, uma vez que nas matrizes *3x3* iriam provocar um transbordo na memória ou iriam consumir demasiado tempo a executar.

No geral, apesar do projeto não responder a todas as questões do enunciado, permitiu ao grupo adquirir conhecimentos sobre a construção e manipulação de estruturas no *Octave*, permitindo pôr em prática a teoria sobre a caracterização de todas as estruturas magmas até nove elementos.

4. BIBLIOGRAFIA

N. Martins-Ferreira, M. Belbut, *Guide To The Project on the Classification of Magmas and Smigroup-Like Structures*, CDRSP-IPLeiria Technical Report, GTLab(Monoids-9.1) 115 (2022)