|  |  |
| --- | --- |
|  |  |
| Toolbox Octave Estrutura Multi-Link |  |
|  |  |
|  | Matemática Discreta |
|  | Grupo Yellow |

1 – function [u, r, q] = unique(A, ”rows”)

A picture containing text, font, screenshot, line

Description automatically generated

Dado que esta função se apresenta como parte do octave, restou-nos entender o algoritmo ‘por detrás’ da função em si. Tendo um conjunto de coordenadas em Rn, indexadas pelo conjunto A, esta função devolve-nos u, as coordenadas únicas em Rn indexadas; q, os índices únicos de A indexados; e r, os índices únicos mapeados á ordenação original.

#project values

A = [1:8]';

f = [1+1i;1+4i;4+4i;4+1i;5+4i;4+4i;4+1i;5+1i];

g = [1+4i;4+4i;4+1i;1+1i;4+4i;4+1i;5+1i;5+4i];

>> [u\_source,r\_source,q\_source]=unique(f,"rows")

u\_source =

   1 + 1i

   4 + 1i

   1 + 4i

   5 + 1i

   4 + 4i

   5 + 4i

r\_source =

   1

   4

   2

   8

   3

   5

q\_source =

   1

   3

   5

   2

   6

   5

   2

   4

>> [u\_target,r\_target,q\_target]=unique(g,"rows")

u\_target =

   1 + 1i

   4 + 1i

   1 + 4i

   5 + 1i

   4 + 4i

   5 + 4i

r\_target =

   4

   3

   1

   7

   2

   8

q\_target =

   3

   5

   2

   1

   5

   2

   4

   6

2 – function [s, t, u] = digraph(source, target)

A picture containing text, screenshot, font, line

Description automatically generated

Esta função realiza a transformação dos dados de fontes e alvos em uma representação adequada para a construção de um dígrafo. As matrizes de fontes e alvos são combinadas em uma única matriz, as linhas duplicadas são removidas para obter uma matriz única e, em seguida, os índices das fontes e alvos são extraídos para formar os vetores s e t, respetivamente.

Explicação algorítmica:

1. A função concatena verticalmente as matrizes de fontes e alvos em uma matriz chamada M.
2. Em seguida, encontra as linhas únicas na matriz M usando a função unique, armazenando as linhas únicas na matriz u e os índices correspondentes em r.
3. O número de linhas na matriz de fontes (source) é armazenado na variável n.
4. Os primeiros n elementos do vetor de índices q são atribuídos ao vetor s, representando os índices das fontes no dígrafo.
5. Em seguida, os n elementos iniciais são removidos do vetor q.
6. O vetor t é atribuído ao vetor q, representando os índices dos alvos no dígrafo.
7. Opcionalmente, o vetor u pode ser retornado como saída da função.

*function* [*s, t, u*] = digraph(source, target)

    % Função para criar um digrafo a partir de uma matriz de fontes (source) e alvos (target).

    % Entrada: source - vetor com as fontes do digrafo

    %          target - vetor com os alvos do digrafo

    % Saída: s - vetor com os índices das fontes no digrafo

    %         t - vetor com os índices dos alvos no digrafo

    %         u - matriz única contendo todas as fontes e alvos

    try

        M = [source; target]; % Concatena as matrizes source e target verticalmente

    catch ME

        error(ME.message) % Se ocorrer um erro, exibe a mensagem de erro

    end

    [u, r, q] = unique(M, 'rows'); % Encontra as linhas únicas na matriz M

    n = size(source, 1); % Obtém o número de linhas na matriz source

    s = q(1:n); % Índices das fontes no digrafo

    q(1:n) = []; % Remove os índices das fontes do vetor q

    t = q; % Índices dos alvos no digrafo

    % A função não retorna o vetor u, mas você pode descomentar a linha abaixo se quiser retorná-lo.

    % u = unique(M, 'rows'); % Matriz única contendo todas as fontes e alvos

end

>> [s,t,u\_digraph]=digraph(q\_source,q\_target)

s =

   1

   3

   5

   2

   6

   5

   2

   4

t =

   3

   5

   2

   1

   5

   2

   4

   6

u\_digraph =

   1

   2

   3

   4

   5

   6

3 – function phi = linkdigraph(A, s, t)

A picture containing text, screenshot, font, document

Description automatically generated

A função linkdigraph recebe uma matriz de adjacência A de um grafo direcionado, juntamente com dois vetores s e t que representam as fontes e os destinos das arestas. O objetivo dessa função é construir uma função endofunção phi que mapeia os nós de destino para os nós de origem. O grupo produziu uma função que funciona através do algoritmo da inversa que segue em anexo e uma nova com o parâmetro A.

Explicação algorítmica:

1. A função começa criando um novo grafo direcionado usando os vetores s e t, juntamente com a matriz de adjacência A. Isso é feito usando a função digraph do MATLAB, que retorna os vetores s, t e u, onde u representa os pesos das arestas (que não são usados neste caso).
2. Em seguida, a função tenta criar uma matriz M concatenando os vetores s e t. Se ocorrer algum erro durante essa concatenação, o erro é capturado e uma mensagem de erro é exibida.
3. A função determina o comprimento do vetor s (que é igual ao número de nós no grafo) e cria um mapa inverso chamado sinv. O mapa inverso é usado para mapear cada valor em s para o seu índice correspondente no vetor.
4. Finalmente, a função retorna o vetor phi, que é construído mapeando cada valor em t para o seu índice correspondente usando o mapa inverso sinv.

function phi = linkdigraph(A, s, t)

[s,t,u] = digraph(s(A), t(A));

try

    M=[s, t];

catch ME

    error(ME.message);

end

    n = length(s);

    sinv(s)=(1:n);

    phi = (sinv(t));

end

>>phi=linkdigraph(A,s,t)

phi *=*

   2

   6

   7

   1

   6

   7

   8

   5

4 – function h = linkdraw(A, phi, g, varargin)

A picture containing text, font, screenshot, line

Description automatically generated

Esta função relaciona elementos do conjunto A usando a função phi e o vetor g. Ela visualiza essas relações como setas no plano complexo, onde cada seta vai de g(A) até g(phi(A)). Além disso, ela permite opcionalmente a visualização de ângulos entre setas consecutivas, usando o valor de v como parâmetro para determinar o tamanho dos arcos.

Explicação algorítmica:

1. O conjunto A é convertido em um vetor coluna.
2. Os vetores phi e g são restringidos ao domínio A.
3. É calculado o vetor g(phi(A)) que representa as extremidades das setas.
4. É calculado o vetor direção das setas subtraindo g(A) de g(phi(A)).
5. O ponto médio das setas é calculado como a média entre g(A) e g(phi(A)).
6. As componentes real e imaginária de g(A) são armazenadas nas variáveis x e y, respectivamente.
7. As componentes real e imaginária do vetor direção são armazenadas nas variáveis u e v, respectivamente.
8. A função quiver é usada para desenhar as setas no plano.
9. Se o número de argumentos de entrada for maior que 3, é realizado um procedimento adicional:
   1. O vetor lambdaA é extraído dos argumentos de entrada.
   2. O texto com os valores de lambdaA é colocado nos pontos médios das setas.
10. Se o número de argumentos de entrada for maior que 4, é realizado um procedimento adicional:
    1. O valor v é extraído dos argumentos de entrada.
    2. São realizados cálculos para desenhar ângulos entre setas consecutivas.
    3. Os ângulos são desenhados como arcos anexados às setas e suas setas sucessivas.
    4. Os valores dos ângulos são exibidos próximos aos arcos.
    5. O gráfico resultante é ajustado para ter escala igual nos eixos x e y

*function* *h* = linkdraw(A, phi, g, varargin)

    % h = linkdraw(A, phi, g, lambda)

    %

    % phi, g e lambda são vetores de tamanho igual

    % A é um subconjunto dos números naturais usado como índices

    % cada elemento de phi(A) deve estar em A

    %

    % Exemplo:

    % A = 1:7; lambda = A;

    % g = [1, 3+i, 5, 3+6i, 4+2i, 2+2i, 3+4i];

    % phi = [2 3 4 1 6 7 5];

    %

    % Se varargin for um número entre 0 e 1, desenha o círculo que ilustra os ângulos entre as setas consecutivas

    %

    % Exemplo:

    % t = .55; A = [1 2 3 4 5 6]; phi = [2 3 4 5 6 1]; g = [0 1 t\*(1+0.5i) 1+1i .25+.75i 1i]; lambda = [1 2 3 4 5 6];

    % linkdraw(A, phi, g, lambda, .15);

    A = A(:); % garante que A seja um vetor coluna

    try

        phiA = phi(A); % restringe phi ao domínio A

        gA = g(A); % restringe g ao domínio A

        gphiA = g(phiA);

        %lambdaA = lambda(A);

    catch ME

        error(ME.message)

    end

    dirA = gphiA - gA;

    midA = (gA + gphiA) / 2;

    x = real(gA);

    y = imag(gA);

    u = real(dirA);

    v = imag(dirA);

    h = quiver(x, y, u, v, 0);

    %---------------------------------------------------------------

    % este código é executado se o parâmetro lambda = varargin{1} for fornecido

    if nargin > 3

        lambdaA = varargin{1};

        text(real(midA), imag(midA), num2str(lambdaA(:)))

    end

    % ------------------------------------------------------------

    % este código é executado apenas se 5=3+2 parâmetros forem fornecidos

    if nargin > 4

        v = varargin{2};

        center = g(phi(A(:)));

        incom = g(A(:)) - center;

        outcom = g(phi(phi(A(:)))) - center;

        radius = v \* mean(abs([incom(:); outcom(:)]));

        ang\_begin = angle(incom); % onde o ângulo começa

        ang\_amp = angle(outcom ./ incom); % amplitude

        ang\_amp(ang\_amp < 0) = ang\_amp(ang\_amp < 0) + 2 \* pi; % transforma em [0 2pi[

        dg = ((ang\_amp / pi) \* 180); % transforma em graus vezes 10 (0--36) em vez de (0--360)

        disp([A(:), ang\_begin(:), ang\_amp(:), dg(:)])

        dg = round(dg);

        hold on

        % desenha os ângulos (como arco com raio v - o parâmetro fornecido) anexado a cada seta e sua seta sucessiva

        for j = 1:numel(A)

            z0 = center(j);

            t = linspace(0, 1, abs(dg(j)) / 10);

            angpoints = radius \* exp((t \* (ang\_amp(j)) + ang\_begin(j)) \* sqrt(-1));

            plot(z0 + angpoints, '.')

            if abs(dg) > 5

                tx = z0 + angpoints(round(((end + 1) / 2))); % centro+v\*exp((0.5\*(ang\_amp)+ang\_begin)\*sqrt(-1));

                text(real(tx), imag(tx), num2str(dg(j)))

            end

        end

        axis equal

        hold off

    end

end

>> h=linkdraw(A,phi,g)

h = -39.255

A picture containing line, diagram, plot, design

Description automatically generated

5 – function [orb, ord, psi] = orbits(phi)

A picture containing text, font, screenshot, handwriting

Description automatically generated

A função phi representa uma relação entre elementos de um conjunto A. A função orbits realiza uma análise dessa relação, obtendo a função q (orb) que mapeia elementos de A para índices em um conjunto B, as órbitas disjuntas que agrupam os elementos relacionados, a ordenação dentro de cada órbita e a função inversa à esquerda (psi) que mapeia elementos de A para seus índices correspondentes em A. Essa função auxilia na compreensão das propriedades das órbitas e da estrutura da relação definida por phi.

Explicação algorítmica:

1. Obtém os elementos únicos de A, que é o domínio da função phi.
2. Define o conjunto B como uma sequência de números inteiros de 1 até o tamanho de A.
3. Define a função orb usando uma função anônima que mapeia cada elemento x de A para o índice correspondente em B.
4. Mapeia cada elemento de A para o índice correspondente em B usando a função orb. Isso determina a órbita a qual cada elemento pertence.
5. Separa os índices em células de acordo com a órbita correspondente.
6. Obtém as órbitas correspondentes como células contendo os elementos de A.
7. Ordena os elementos dentro de cada órbita.
8. Define a função psi usando uma função anônima que mapeia cada elemento x de A para o índice correspondente em A.

*function* [*orb, ord, psi*] = orbits(phi)

    % Função para obter a função q, as órbitas, a ordenação dentro de cada órbita e a função inversa à esquerda para phi.

    % Entrada: phi - endofunção phi: A -> A

    % Saída: orb - função q: A -> B

    %        ord - ordenação dentro de cada órbita disjunta

    %        psi - função inversa à esquerda para phi

    A = unique(phi); % Obtém os elementos únicos de A

    B = 1:length(A); % Define o conjunto B como uma sequência de números inteiros de 1 até o tamanho de A

    % Passo 1: Definir a função q: A -> B

    orb = @(x) find(A == phi(x)); % Mapeia cada elemento x de A para o índice correspondente em B

    % Passo 2: Obter as órbitas disjuntas

    orb\_indices = arrayfun(orb, A); % Mapeia cada elemento de A para o índice correspondente em B

    orb\_cells = arrayfun(@(x) find(orb\_indices == x), 1:length(A), 'UniformOutput', false); % Separa os índices em células de acordo com a órbita correspondente

    orb = cellfun(@(x) A(x), orb\_cells, 'UniformOutput', false); % Obtém as órbitas correspondentes como células contendo os elementos de A

    % Passo 3: Obter a ordenação dentro de cada órbita disjunta

    ord = cellfun(@(x) sort(x), orb, 'UniformOutput', false); % Ordena os elementos dentro de cada órbita

    % Passo 4: Obter a função inversa à esquerda para phi

    psi = @(x) find(A == x); % Mapeia cada elemento x de A para o índice correspondente em A

end

>> [orb,ord,psi]=orbits(phi)

orb *=*

{

  [1,1] = 4

  [1,2] = 1

  [1,3] = 2

  [1,4] = 3

  [1,5] = 8

  [1,6] = 5

  [1,7] = 6

  [1,8] = 7

}

ord *=*

{

  [1,1] = 4

  [1,2] = 1

  [1,3] = 2

  [1,4] = 3

  [1,5] = 8

  [1,6] = 5

  [1,7] = 6

  [1,8] = 7

}

psi *=*

@(x) find (A == x)

6 – function [phi, orb, ord, psi] = orbitsgen(n, k, ck, pk)

A picture containing text, screenshot, font, line

Description automatically generated

Esta função gera órbitas de componentes com base nos parâmetros fornecidos. Ela retorna as matrizes phi, orb, ord e psi. Em termos matemáticos, uma órbita é um conjunto de elementos relacionados por uma função. A função orbitsgen constrói órbitas com base em componentes cíclicos, principais e secundários. Cada componente é um conjunto de elementos que são relacionados entre si.

Explicação algorítmica:

1. Define funções auxiliares que geram componentes cíclicos, principais e secundários.
2. Inicializa as matrizes phi, orb, ord e psi com zeros.
3. Divide o conjunto n em k componentes usando a função partition.
4. Separa os componentes não cíclicos e cíclicos a partir da partição.
5. Constrói os componentes principais: adiciona cada componente principal à matriz phi, orb e psi, atribui valores às matrizes ord e orb para representar a ordem dos elementos dentro das órbitas.
6. Constrói os componentes secundários: adiciona cada componente secundário à matriz phi, orb e psi, atualiza as matrizes ord e orb para representar a ordem dos elementos dentro das órbitas.
7. Constrói os componentes cíclicos: adiciona cada componente cíclico à matriz phi, orb e psi, atualiza as matrizes ord e orb para representar a ordem dos elementos dentro das órbitas.
8. Retorna as matrizes phi, orb, ord e psi.

*function* [*phi, orb, ord, psi*] = orbitsgen(n, k, ck, pk)

% [phi, orb, ord, psi] = orbitsgen(n, k, ck, pk)

%

% Esta função gera órbitas de componentes com base nos parâmetros fornecidos.

% Retorna as matrizes phi, orb, ord e psi.

%

% Exemplo:

% n = 20, k = 7, ck = 2, pk = 2,

% [phi, orb, ord, psi] = orbitsgen(n, k, ck, pk);

% Funções auxiliares para gerar componentes cíclicos, principais e secundários

u = @(x) [2:x, 1]';                  % Componente cíclico de comprimento x

v = @(x) [2:x, randi([2, x])]';      % Componente principal de comprimento x

w = @(x) [2:x, NaN]';                % Componente secundário com último elemento a ser determinado

id = @(x) (1:x)';

ur = @(x) [x, 1:x-1]';               % Componente cíclico de comprimento x, invertido

vr = @(x) [0, 1:x-1]';               % Componente principal de comprimento x, invertido

wr = @(x) [0, 1:x-1]';               % Componente secundário com último elemento a ser determinado, invertido

phi = zeros(n, 1);

orb = zeros(n, 1);

ord = zeros(n, 1);

psi = zeros(n, 1);

% Vamos assumir que os últimos ck componentes em p são os componentes cíclicos,

% os primeiros cp são os componentes principais, e vamos assumir que os

% componentes principais são mais longos do que os secundários, então:

p = partition(n, k);

p\_ncyc = sort(p(1:end-ck), 'descend');     % Componentes não cíclicos de p

p\_cyc = p(end-ck+1:end);                    % Componentes cíclicos

% Construção dos componentes principais

nt = 0;  % Comprimento atual do vetor phi com os componentes principais adicionados

for i = 1:pk

    x = p\_ncyc(i);

    orb(nt+id(x)) = -i;

    ord(nt+id(x)) = id(x);

    ord(nt+x) = Inf;

    phi(nt+id(x)) = nt+v(x);

    psi(nt+id(x)) = nt+vr(x);

    psi(nt+1) = 0;  % O pseudo-inverso de phi

    nt = nt + x;

end

% Componentes secundários

ntt = 0;  % ntt é o comprimento atual, enquanto nt é o comprimento total dos componentes principais

idnt = (1:nt)';

idnt = idnt(psi(1:nt) ~= 0);

for i = 1+pk:k-ck

    x = p\_ncyc(i);

    ri = idnt(randi(nt-pk));

    orb(nt+ntt+id(x)) = -i;

    ord(nt+ntt+id(x)) = id(x);

    ord(nt+ntt+x) = orb(ri);

    phi(nt+ntt+id(x)) = [nt+ntt+[2:x], ri];

    psi(nt+ntt+id(x)) = nt+ntt+vr(x);

    psi(nt+ntt+1) = 0;  % O pseudo-inverso de phi

    ntt = ntt + x;

end

% Componentes cíclicos

nttt = 0;  % ntt é o comprimento atual, enquanto nt é o comprimento total dos componentes principais

nt = nt + ntt;  % Não há mais necessidade de manter a distinção entre nt e ntt

for i = 1:ck

    x = p\_cyc(i);

    orb(nt+nttt+id(x)) = i;

    ord(nt+nttt+id(x)) = id(x);

    phi(nt+nttt+id(x)) = nt+nttt+[2:x, 1];

    psi(nt+nttt+id(x)) = nt+nttt+ur(x);  % O pseudo-inverso de phi

    nttt = nttt + x;

end

7 – function g = orbitsviz(phi, orb, ord)

A picture containing text, screenshot, font, line

Description automatically generated

Esta função, orbitsviz, realiza a visualização das órbitas no plano complexo. Ela recebe como entrada três matrizes: phi, orb e ord. A matriz phi representa a endofunção phi: A -> A, onde A é um conjunto de elementos. A matriz orb contém as órbitas correspondentes a phi, e a matriz ord contém a ordenação dentro de cada órbita. Dessa forma, a função percorre cada elemento de phi, verifica se está em uma órbita e calcula a posição correspondente no plano complexo com base nas informações das órbitas e ordenações fornecidas. O resultado é armazenado no vetor g, que contém a visualização das órbitas no plano complexo. **Não conseguimos por esta função a produzir o output desejado.**

Explicação algorítmica:

1. Verificar o tamanho das matrizes de entrada para garantir que tenham o mesmo tamanho.
2. Criar um vetor g para armazenar a visualização no plano complexo. Inicialmente, o vetor é preenchido com zeros.
3. Percorrer cada elemento da matriz phi.
4. Obter o valor do elemento atual, denotado por x.
5. Verificar se o elemento está dentro de uma órbita. Isso é feito comparando o valor correspondente em orb com zero.
6. Se o elemento estiver dentro de uma órbita:
   1. Obter a posição do elemento dentro da órbita, encontrando o índice em ord que corresponde a x.
   2. Calcular a posição no plano complexo usando ângulo e distância.
      1. O ângulo é calculado como 2 \* pi \* (pos / numel(ord(orb(i), :))), onde pos é a posição do elemento na órbita e numel(ord(orb(i), :)) é o número total de elementos na órbita.
      2. O raio é definido como o valor da órbita orb(i).
   3. Armazenar o valor correspondente no vetor g usando a fórmula radius \* exp(1i \* angle), onde 1i é a unidade imaginária.
7. Retornar o vetor g, que representa a visualização das órbitas no plano complexo.

*function* *g* = orbitsviz(phi, orb, ord)

    % Função para visualização das órbitas no plano complexo

    % Entrada: phi - matriz de endofunção phi: A -> A

    %          orb - matriz de órbitas correspondentes a phi

    %          ord - matriz de ordenação dentro de cada órbita

    % Saída: g - vetor que representa a visualização no plano complexo

    % Verificar o tamanho das matrizes de entrada

    [n, m] = size(phi);

    if ~isequal(size(orb), [n, m]) || ~isequal(size(ord), [n, m])

        error('As matrizes de entrada devem ter o mesmo tamanho.')

    end

    % Criar um vetor g para armazenar a visualização no plano complexo

    g = zeros(n, 1);

    % Percorrer cada elemento da matriz phi

    for i = 1:n

        % Obter o valor do elemento atual

        x = phi(i);

        % Verificar se o elemento está dentro de uma órbita

        if orb(i) ~= 0

            % Obter a posição do elemento dentro da órbita

            pos = find(ord(orb(i), :) == x);

            % Calcular a posição no plano complexo usando ângulo e distância

            angle = 2 \* pi \* (pos / numel(ord(orb(i), :)));

            radius = orb(i);

            g(i) = radius \* exp(1i \* angle);

        end

    end

end

8 – function gx = linkoffset(A, phi, g, x)

A picture containing text, screenshot, font, line

Description automatically generated

A função linkoffset permite calcular o deslocamento de uma região representada por g em relação à região original, aplicando o offset x aos elementos de A antes de calcular a função g. O resultado é uma nova função gx que representa a região deslocada. **Não conseguimos por esta função a produzir o output desejado.**

Explicação algorítmica:

1. Verificar se o tamanho de A, phi e g é compatível. Isso é feito comparando os tamanhos das matrizes. Caso não sejam compatíveis, é lançado um erro indicando que os tamanhos devem ser os mesmos.
2. Definir a função phi\_offset, que calcula a função phi com o offset x aplicado. Para cada elemento a em A, a função phi\_offset(a) é definida como o resultado de phi(a) somado ao valor de x.
3. Definir a função gx, que aplica o offset x à função g. Para cada elemento a em A, a função gx(a) é definida como o resultado de g(phi\_offset(a)), ou seja, o valor de g aplicado à imagem de a sob a função phi\_offset.
4. Retornar a função gx, que representa a região deslocada pelo offset x em relação à região original.

*function* *gx* = linkoffset(A, phi, g, x)

    % Função para calcular o deslocamento (offset) de uma região representada por g em relação à região original

    % Entrada: A - conjunto de elementos A

    %          phi - endofunção phi: A -> A

    %          g - função g: A -> C (números complexos)

    %          x - offset a ser aplicado à região original

    % Saída: gx - função g com o deslocamento aplicado

    % Verificar se o tamanho de A, phi e g são compatíveis

    if ~isequal(size(A), size(phi)) || ~isequal(size(A), size(g))

        error('O tamanho de A, phi e g deve ser o mesmo.')

    end

    % Calcular a função phi com o offset aplicado

    phi\_offset = @(a) phi(a) + x;

    % Aplicar o offset à função g

    gx = @(a) g(phi\_offset(a));

end

9 – função interpolação

A picture containing text, font, screenshot, number

Description automatically generated

**O grupo não conseguiu produzir qualquer resultado para esta função.**

10 – função isoslice



**O grupo não conseguiu produzir qualquer resultado para esta função.**

*Créditos:*

*João Parreira*

*Francisco Francisco*

*André Rosa*

*Bruno Marcelino*

*Jorge Gonçalves*