

# Projekt: Das Problem der 36 Offiziere

Ilka Schiermeier & Julia Hoffrichter

22.01.2003

Auf einem Regimentsball treffen sich 36 Offiziere aus 6 verschiedenen Regimenten mit 6 unterschiedlichen Dienstgraden, wobei jeder Dienstgrad in jedem Regiment auftritt. Zur Feier des Tages sollen diese 36 Offiziere eine kleine Parade machen, aber zur Abwechslung mal nicht säuberlich geordnet, sondern gerade entgegengesetzt sodass in jeder Zeile und Spalte eines  $6 \times 6$ -Quadrates aus jedem Regiment und von jedem Dienstgrad (genau) ein Offizier dabei ist.

## Definition:

Eine  $n \times n$ -Matrix mit Einträgen in  $\{1, \dots, n\}$  nennen wir **wim-Quadrat**.

Ein  $n$ -Quadrat heißt **lateinisches Quadrat** genau dann, wenn jeder Eintrag in jeder Zeile und in jeder Spalte genau einmal vorkommt.

Zwei Quadrate heißen **orthogonal**, wenn beim Übereinanderlegen jedes (geordnete) Paar genau einmal entsteht.

Ein **graeco-lateinisches Quadrat** ist ein orthogonales Paar von lateinischen Quadraten.

Ein **normiertes Quadrat** nennt man ein Quadrat, dessen erste Zeile 1, 2, 3, ...,  $n$  lautet.

Zuerst haben wir eine Prozedur entwickelt, die testet, ob ein Quadrat lateinisch ist.

```
> restart:
with(LinearAlgebra):
with(combinat, permute):
> islatin:=proc(A)
local merker,size,garbage,i,j,k;
size,garbage := op(1,A);
merker := Vector(size);
for i from 1 to size do
  for k from 1 to size do
    merker[k] := 0;
  end do;
  for j from 1 to size do
    if merker[A[i,j]] = 1 then
      return false
    else
      merker[A[i,j]]:= 1;
    end if;
  end do;
end do;

for i from 1 to size do
  for k from 1 to size do
    merker[k] := 0;
  end do;
```

```

for j from 1 to size do
  if merker[A[j,i]] = 1 then
    return false
  else
    merker[A[j,i]] := 1;
  end if;
end do;
end do;
return true;
end proc:

```

[ Mal testen, ob die Prozedur funktioniert.

```
> A:=<<1,2>|<2,1>>;
```

$$A := \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$$

```
> islatin(A);
```

*true*

```
> B:=<<2,1>|<2,1>>;
```

$$B := \begin{bmatrix} 2 & 2 \\ 1 & 1 \end{bmatrix}$$

```
> islatin(B);
```

*false*

Das scheint zu stimmen. Unser Ziel ist es nun, eine Prozedur zu entwickeln, die uns alle lateinischen Matrizen des gewünschten Formates liefert. Auf Wunsch sollen die Matrizen normiert werden.

```

> dofindlatin:=proc(size,matrix,actualrow,permutation,sizepermute,
norm)
local i,value,j,L;
if actualrow > size then
  if islatin(matrix) = true then
    print(matrix);
    return copy(matrix);
  end if;
  return NULL;
else
  L := NULL;
  for i from 1 to sizepermute do
    if actualrow = 1 and norm = 1 then
      value := [seq(i,i=1..size)];
    else
      value := permutation[i];
    end if;
    for j from 1 to size do
      matrix[actualrow,j] := value[j];
    end do;
    L :=
L,dofindlatin(size,matrix,actualrow+1,permutation,sizepermute);
    if actualrow = 1 and norm = 1 then
      break;
    end if;
  end do;
end do;

```

```

    return L;
end if;
end proc;
> findlatin:=proc(size,norm)
local permutations,startvector,i,permuteseize,garbage,startmatrix;
permutations := permute([seq(i,i=1..size)]);
permuteseize := nops(permutations);
startmatrix := Matrix(size,size);
return
do findlatin(size,startmatrix,1,permutations,permuteseize,norm);
end proc;

```

[ Nun hätten wir gerne alle (nicht unbedingt normierten) 3 x 3-Matrizen.

```
> findlatin(3,0);
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \\ 2 & 3 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 & 2 \\ 2 & 1 & 3 \\ 3 & 2 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 & 2 \\ 3 & 2 & 1 \\ 2 & 1 & 3 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 1 & 3 \\ 1 & 3 & 2 \\ 3 & 2 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 1 & 3 \\ 3 & 2 & 1 \\ 1 & 3 & 2 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 3 & 1 \\ 1 & 2 & 3 \\ 3 & 1 & 2 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 3 & 1 \\ 3 & 1 & 2 \\ 1 & 2 & 3 \\ 3 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \\ 2 & 3 & 1 \\ 1 & 2 & 3 \\ 3 & 2 & 1 \\ 1 & 3 & 2 \\ 2 & 1 & 3 \\ 3 & 2 & 1 \\ 2 & 1 & 3 \\ 1 & 3 & 2 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{bmatrix}, \begin{bmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \\ 2 & 3 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 3 & 2 \\ 2 & 1 & 3 \\ 3 & 2 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 3 & 2 \\ 3 & 2 & 1 \\ 2 & 1 & 3 \end{bmatrix}, \begin{bmatrix} 2 & 1 & 3 \\ 1 & 3 & 2 \\ 3 & 2 & 1 \end{bmatrix}, \begin{bmatrix} 2 & 1 & 3 \\ 3 & 2 & 1 \\ 1 & 3 & 2 \end{bmatrix},$$

$$\begin{bmatrix} 2 & 3 & 1 \\ 1 & 2 & 3 \\ 3 & 1 & 2 \end{bmatrix}, \begin{bmatrix} 2 & 3 & 1 \\ 3 & 1 & 2 \\ 1 & 2 & 3 \end{bmatrix}, \begin{bmatrix} 3 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 1 \end{bmatrix}, \begin{bmatrix} 3 & 1 & 2 \\ 2 & 3 & 1 \\ 1 & 2 & 3 \end{bmatrix}, \begin{bmatrix} 3 & 2 & 1 \\ 1 & 3 & 2 \\ 2 & 1 & 3 \end{bmatrix}, \begin{bmatrix} 3 & 2 & 1 \\ 2 & 1 & 3 \\ 1 & 3 & 2 \end{bmatrix}$$

Eigentlich interessieren uns aber nur die normierten, also ändern wir unsere Eingabe.

```
> findlatin(3,1);
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \\ 1 & 2 & 3 \\ 3 & 1 & 2 \\ 2 & 3 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{bmatrix}, \begin{bmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \\ 2 & 3 & 1 \end{bmatrix}$$

```
> findlatin(4,1);
```

```
[ 1  2  3  4]
[ 2  1  4  3]
[ 3  4  1  2]
[ 4  3  2  1]
[ 1  2  3  4]
[ 2  1  4  3]
[ 3  4  2  1]
[ 4  3  1  2]
[ 1  2  3  4]
[ 2  1  4  3]
[ 4  3  1  2]
[ 3  4  2  1]
[ 1  2  3  4]
[ 2  3  4  1]
[ 3  4  1  2]
[ 4  1  2  3]
[ 1  2  3  4]
[ 2  3  4  1]
[ 4  1  2  3]
[ 3  4  1  2]
[ 1  2  3  4]
[ 2  4  1  3]
[ 3  1  4  2]
[ 4  3  2  1]
```

[ 1 2 3 4 ]  
[ 2 4 1 3 ]  
[ 4 3 2 1 ]  
[ 3 1 4 2 ]  
[ 1 2 3 4 ]  
[ 3 1 4 2 ]  
[ 2 4 1 3 ]  
[ 4 3 2 1 ]  
[ 1 2 3 4 ]  
[ 3 1 4 2 ]  
[ 4 3 2 1 ]  
[ 2 4 1 3 ]  
[ 1 2 3 4 ]  
[ 3 4 1 2 ]  
[ 2 1 4 3 ]  
[ 4 3 2 1 ]  
[ 1 2 3 4 ]  
[ 3 4 1 2 ]  
[ 2 3 4 1 ]  
[ 4 1 2 3 ]  
[ 1 2 3 4 ]  
[ 3 4 1 2 ]  
[ 4 1 2 3 ]  
[ 2 3 4 1 ]  
[ 1 2 3 4 ]  
[ 3 4 1 2 ]  
[ 4 3 2 1 ]  
[ 2 1 4 3 ]

[ 1 2 3 4 ]  
[ 3 4 2 1 ]  
[ 2 1 4 3 ]  
[ 4 3 1 2 ]

[ 1 2 3 4 ]  
[ 3 4 2 1 ]  
[ 4 3 1 2 ]  
[ 2 1 4 3 ]

[ 1 2 3 4 ]  
[ 4 1 2 3 ]  
[ 2 3 4 1 ]  
[ 3 4 1 2 ]

[ 1 2 3 4 ]  
[ 4 1 2 3 ]  
[ 3 4 1 2 ]  
[ 2 3 4 1 ]

[ 1 2 3 4 ]  
[ 4 3 1 2 ]  
[ 2 1 4 3 ]  
[ 3 4 2 1 ]

[ 1 2 3 4 ]  
[ 4 3 1 2 ]  
[ 3 4 2 1 ]  
[ 2 1 4 3 ]

[ 1 2 3 4 ]  
[ 4 3 2 1 ]  
[ 2 1 4 3 ]  
[ 3 4 1 2 ]



$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 4 & 1 & 2 & 3 \\ 2 & 3 & 4 & 1 \\ 3 & 4 & 1 & 2 \end{bmatrix}
 \begin{bmatrix} 1 & 2 & 3 & 4 \\ 4 & 1 & 2 & 3 \\ 3 & 4 & 1 & 2 \\ 2 & 3 & 4 & 1 \end{bmatrix}
 \begin{bmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 1 & 2 \\ 2 & 1 & 4 & 3 \\ 3 & 4 & 2 & 1 \end{bmatrix}
 \begin{bmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 1 & 2 \\ 3 & 4 & 2 & 1 \\ 2 & 1 & 4 & 3 \end{bmatrix}$$

Wir wollen jetzt herausfinden, ob zwei Quadrate orthogonal sind. Dazu brauchen wir eine Hilfsprozedur, die zwei Quadrate übereinander legt. Die eigentliche Prozedur prüft dann nur, ob jeder Eintrag genau ein Mal vorkommt.

```

> mergeMatrices:=proc(A,B)
  local endmatrix,i,j,element,size,garbage;
  size,garbage := op(1,A);
  for i from 1 to size do
    for j from 1 to size do
      element := Vector(2);
      element[1] := A[i,j];
      element[2] := B[i,j];
      endmatrix[i,j] := element;
    end do;
  end do;
  eval(endmatrix);
end proc;

> isorthogonal:=proc(A,B)
  local mergematrix,size,garbage,merker,i,j,value;
  size,garbage := op(1,A);
  mergematrix := mergeMatrices(A,B);
  merker := table();
  for i from 1 to size do
    for j from 1 to size do
      value := op(2,mergematrix[i,j]);
      if merker[value] = 1 then
        return false;
      else
        merker[value] := 1;
      end if;
    end do;
  end do;
  return true;
end proc;

```

Ein erneuter Test!

```
> A:=<<1,2>|<2,1>>;
```

$$A := \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$$

```
> B:=<<2,1>|<1,2>>;
```

$$B := \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

```
> isorthogonal(A,B);
```

*false*

[ Nun mal ein Test mit zwei Matrizen, die eigentlich orthogonal sein sollten.

```
> C:=<<1,2,3>|<2,3,1>|<3,1,2>>;
```

$$C := \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{bmatrix}$$

```
> E:=<<1,3,2>|<2,1,3>|<3,2,1>>;
```

$$E := \begin{bmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \\ 2 & 3 & 1 \end{bmatrix}$$

```
> isorthogonal (C,E);
```

*true*

Das ist ja schön :))

So, jetzt aber zurück zum Problem der 36 Offiziere.

```
> offiziere:=proc(matrixsize)
  local L,returnsize,i,j,garbage,isort;
  L:=[findlatin(matrixsize,1)];
  returnsize := nops(L);
  for i from 1 to returnsize do
    for j from i+1 to returnsize do
      isort := isortogonal(L[i],L[j]);
      if isort = true then
        print(L[i],L[j],isort);
      end if;
    end do;
  end do;
end proc;
```

```
offiziere := proc(matrixsize)
```

```
local L, returnsize, i, j, garbage, isort;
```

```
  L := [findlatin(matrixsize, 1)];
```

```
  returnsize := nops(L);
```

```
  for i to returnsize do for j from i + 1 to returnsize do
```

```
    isort := isortogonal(L[i], L[j]); if isort = true then print(L[i], L[j], isort) end if
```

```
  end do
```

```
  end do
```

```
end proc
```

```
> offiziere(3);
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \\ 2 & 3 & 1 \end{bmatrix}$$

> `offiziere(2):`

$$\begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$$

>  
Die Prozedur sucht sich einfach alle lateinischen Matrizen einer bestimmten Größe und paart dann jede Matrix mit jeder anderen, um zu schauen, ob diese Paarung zueinander orthogonal ist. Jedes orthogonale Pärchen wird dann ausgegeben.  
!! Achtung !! Prozedur seeeeehr Rechenzeitintensiv, berechnung für 6x6 Matrizen nach 15 mins abgebrochen!