LU-decompositie

Academiejaar 2020-2021

Systeem

```
> restart;
  interface( rtablesize = 15 ):
> with( LinearAlgebra ):
```

Opgave: Los het benchmark stelsel

$$\sum_{j=1}^{n} (1+i)^{j-1} x_j = \frac{(1+i)^n - 1}{i}, \qquad 1 \le i \le n$$

op voor $n = 5, ..., 15$ met naïeve Gauss-eliminatie en ook met partiële pivotering. Bereken daarna de error en het residu en vergelijk de bekomen resultaten met de gekende bovengrenzen.

2.1. Stelsel van lineaire vergelijkingen

Stelsel

```
> n := 5;
```

$$n := 5 \tag{2.1.1}$$

```
> vgl := {seq(add((1+i)^(j-1)*x[j],j = 1..n) = ((1+i)^n-1)/i, i=
  1..n)}:
  print~(vgl)[];
```

$$x_1 + 2\,x_2 + 4\,x_3 + 8\,x_4 + 16\,x_5 = 31$$

$$x_1 + 3\,x_2 + 9\,x_3 + 27\,x_4 + 81\,x_5 = 121$$

$$x_1 + 4\,x_2 + 16\,x_3 + 64\,x_4 + 256\,x_5 = 341$$

$$x_1 + 5\,x_2 + 25\,x_3 + 125\,x_4 + 625\,x_5 = 781$$

$$x_1 + 6\,x_2 + 36\,x_3 + 216\,x_4 + 1296\,x_5 = 1555 \tag{2.1.2}$$

```
> A := n -> Matrix( n, (i,j) -> (1+i)^(j-1) ):
  An := A(5);
```

$$An := \begin{bmatrix} 1 & 2 & 4 & 8 & 16 \\ 1 & 3 & 9 & 27 & 81 \\ 1 & 4 & 16 & 64 & 256 \\ 1 & 5 & 25 & 125 & 625 \\ 1 & 6 & 36 & 216 & 1296 \end{bmatrix} \tag{2.1.3}$$

```
> y := n -> Vector( n, (i) -> ((1+i)^n-1)/i ):
  yn := y(n);
```

$$yn := \begin{bmatrix} 31 \\ 121 \\ 341 \\ 781 \\ 1555 \end{bmatrix} \tag{2.1.4}$$

We moeten het stelsel $A \cdot x = y$ oplossen.

```
> print( 'A.x = y' );
  xn := Vector( n, (i) -> evaln(x[i]) ):
  print( 'A' = An, 'x' = xn, 'y' = yn );
```

$$A \cdot x = y$$

$$A = \begin{bmatrix} 1 & 2 & 4 & 8 & 16 \\ 1 & 3 & 9 & 27 & 81 \\ 1 & 4 & 16 & 64 & 256 \\ 1 & 5 & 25 & 125 & 625 \\ 1 & 6 & 36 & 216 & 1296 \end{bmatrix}, x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}, y = \begin{bmatrix} 31 \\ 121 \\ 341 \\ 781 \\ 1555 \end{bmatrix} \tag{2.1.5}$$

Deze coëfficiëntenmatrix is beter gekend als de **Vandermonde-matrix** $V_{i,j} = v_i^{j-1}$:

```
> V := VandermondeMatrix( v, n );
```

$$V := \begin{bmatrix} 1 & v_1 & v_1^2 & v_1^3 & v_1^4 \\ 1 & v_2 & v_2^2 & v_2^3 & v_2^4 \\ 1 & v_3 & v_3^2 & v_3^3 & v_3^4 \\ 1 & v_4 & v_4^2 & v_4^3 & v_4^4 \\ 1 & v_5 & v_5^2 & v_5^3 & v_5^4 \end{bmatrix} \tag{2.1.6}$$

```
> An := VandermondeMatrix( [$2..(n+1)] );
```

$$An := \begin{bmatrix} 1 & 2 & 4 & 8 & 16 \\ 1 & 3 & 9 & 27 & 81 \\ 1 & 4 & 16 & 64 & 256 \\ 1 & 5 & 25 & 125 & 625 \\ 1 & 6 & 36 & 216 & 1296 \end{bmatrix} \tag{2.1.7}$$

Oplossing

In het geval van een benchmark probleem willen we natuurlijk de oplossing kennen om het gedrag van algortimes te onderzoeken.

We kunnen makkelijk inzien dat de oplossing voor dit stelsel van lineaire vergelijkingen bestaat uit $x_j = 1, j = 1,.., n$.

```
> unassign('n'):
  print( 'sum((1+i)^(j-1),j=1..n)' = simplify(sum((1+i)^(j-1),j=
  1..n)));
```

$$\sum_{j=1}^{n} (1+i)^{j-1} = \frac{(1+i)^n - 1}{i} \qquad \textbf{(2.2.1)}$$

2.6. Naïeve Gauss-eliminatie

Voorwaartse eliminatie

EliminateColumn berekent de matrix $M_k = I + M^{(k)}$ zodat in $U_k = M_k U_{k-1}$ de elementen onder de diagonaal in kolom k geëlimineerd worden.
Op deze manier kunnen we kolom per kolom te werk gaan om een bovendriehoeksmatrix te creëren.

```
> delta := (i,j) -> `if`( i=j, 1, 0 ):
  EliminateColumn := proc( k, Uk )
    local Mk;
    Mk := Matrix( n, (i,j) -> -Uk[i,k]/Uk[k,k]*delta(j,k) );
    Mk[1..k,1..n] := 0;
    return IdentityMatrix(n) + Mk;
  end proc:
```

We voeren dit nu uit voor $k = 1,..., n-1$ met $U_0 = A$.

```
> n := 5:
  An := A(n):
  print( 'U'[0] = An );
  Uk[0] := An:
  for k from 1 to RowDimension(Uk[0])-1 do
    print(```);
    print( M[k].'U'[k-1] = 'U'[k] );
    Mk[k] := EliminateColumn( k, Uk[k-1] ):
    Uk[k] := Mk[k].Uk[k-1]:
    print( M[k] = Mk[k], 'U'[k-1] = Uk[k-1], 'U'[k] = Uk[k]);
  end:
```

$$U_0 = \begin{bmatrix} 1 & 2 & 4 & 8 & 16 \\ 1 & 3 & 9 & 27 & 81 \\ 1 & 4 & 16 & 64 & 256 \\ 1 & 5 & 25 & 125 & 625 \\ 1 & 6 & 36 & 216 & 1296 \end{bmatrix}$$

$$M_1 \cdot U_0 = U_1$$

$$M_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 & 1 \end{bmatrix}, U_0 = \begin{bmatrix} 1 & 2 & 4 & 8 & 16 \\ 1 & 3 & 9 & 27 & 81 \\ 1 & 4 & 16 & 64 & 256 \\ 1 & 5 & 25 & 125 & 625 \\ 1 & 6 & 36 & 216 & 1296 \end{bmatrix}, U_1$$

$$= \begin{bmatrix} 1 & 2 & 4 & 8 & 16 \\ 0 & 1 & 5 & 19 & 65 \\ 0 & 2 & 12 & 56 & 240 \\ 0 & 3 & 21 & 117 & 609 \\ 0 & 4 & 32 & 208 & 1280 \end{bmatrix}$$

$$M_2 \cdot U_1 = U_2$$

$$M_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & -2 & 1 & 0 & 0 \\ 0 & -3 & 0 & 1 & 0 \\ 0 & -4 & 0 & 0 & 1 \end{bmatrix}, U_1 = \begin{bmatrix} 1 & 2 & 4 & 8 & 16 \\ 0 & 1 & 5 & 19 & 65 \\ 0 & 2 & 12 & 56 & 240 \\ 0 & 3 & 21 & 117 & 609 \\ 0 & 4 & 32 & 208 & 1280 \end{bmatrix}, U_2$$

$$= \begin{bmatrix} 1 & 2 & 4 & 8 & 16 \\ 0 & 1 & 5 & 19 & 65 \\ 0 & 0 & 2 & 18 & 110 \\ 0 & 0 & 6 & 60 & 414 \\ 0 & 0 & 12 & 132 & 1020 \end{bmatrix}$$

$$M_3 \cdot U_2 = U_3$$

$$M_3 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -3 & 1 & 0 \\ 0 & 0 & -6 & 0 & 1 \end{bmatrix}, U_2 = \begin{bmatrix} 1 & 2 & 4 & 8 & 16 \\ 0 & 1 & 5 & 19 & 65 \\ 0 & 0 & 2 & 18 & 110 \\ 0 & 0 & 6 & 60 & 414 \\ 0 & 0 & 12 & 132 & 1020 \end{bmatrix}, U_3$$

$$= \begin{bmatrix} 1 & 2 & 4 & 8 & 16 \\ 0 & 1 & 5 & 19 & 65 \\ 0 & 0 & 2 & 18 & 110 \\ 0 & 0 & 0 & 6 & 84 \\ 0 & 0 & 0 & 24 & 360 \end{bmatrix}$$

$$M_4 \cdot U_3 = U_4$$

$$M_4 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -4 & 1 \end{bmatrix}, U_3 = \begin{bmatrix} 1 & 2 & 4 & 8 & 16 \\ 0 & 1 & 5 & 19 & 65 \\ 0 & 0 & 2 & 18 & 110 \\ 0 & 0 & 0 & 6 & 84 \\ 0 & 0 & 0 & 24 & 360 \end{bmatrix}, U_4 = \begin{bmatrix} 1 & 2 & 4 & 8 & 16 \\ 0 & 1 & 5 & 19 & 65 \\ 0 & 0 & 2 & 18 & 110 \\ 0 & 0 & 0 & 6 & 84 \\ 0 & 0 & 0 & 0 & 24 \end{bmatrix} \quad \textbf{(3.1.1)}$$

```
> print( 'U' = foldl( `.`, seq( M[i], i=n-1..1, -1 ), 'A' ) );
  U := foldl( `.`, seq( Mk[i], i=n-1..1, -1 ), An ):
```

$$U = M_4 \cdot M_3 \cdot M_2 \cdot M_1 \cdot A \quad \textbf{(3.1.2)}$$

```
> print( 'L' = foldl( `.`, seq( Inverse(M[i]), i=1..n-1 ) ) );
  L := foldl( `.`, seq( MatrixInverse(Mk[i]), i=1..n-1 ) ):
```

$$L = Inverse(M_1) \cdot Inverse(M_2) \cdot Inverse(M_3) \cdot Inverse(M_4) \quad \textbf{(3.1.3)}$$

```
> print( 'L' = L, 'U' = U );
```

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 \\ 1 & 3 & 3 & 1 & 0 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}, U = \begin{bmatrix} 1 & 2 & 4 & 8 & 16 \\ 0 & 1 & 5 & 19 & 65 \\ 0 & 0 & 2 & 18 & 110 \\ 0 & 0 & 0 & 6 & 84 \\ 0 & 0 & 0 & 0 & 24 \end{bmatrix} \quad \textbf{(3.1.4)}$$

Er geldt: $LU = A$.

```
> Equal( L.U, An );
```

$$true \quad \textbf{(3.1.5)}$$

Achterwaartse substitutie

De oplossing van $A \cdot x = y$ wordt verkregen door het oplossen van het equivalente stelsel $U \cdot x = L^{-1} \cdot y$.

```
> print( 'U.x = Inverse(L).y' );
  print( 'U' = U, 'x' = xn, 'Inverse(L)' = MatrixInverse(L), 'y'
  = yn );
```

$$U \bullet x = Inverse(L) \bullet y$$

$$U = \begin{bmatrix} 1 & 2 & 4 & 8 & 16 \\ 0 & 1 & 5 & 19 & 65 \\ 0 & 0 & 2 & 18 & 110 \\ 0 & 0 & 0 & 6 & 84 \\ 0 & 0 & 0 & 0 & 24 \end{bmatrix}, x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}, Inverse(L) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 \\ -1 & 3 & -3 & 1 & 0 \\ 1 & -4 & 6 & -4 & 1 \end{bmatrix}, y \qquad \textbf{(3.2.1)}$$

$$= \begin{bmatrix} 31 \\ 121 \\ 341 \\ 781 \\ 1555 \end{bmatrix}$$

De oplossing van het stelsel $U \cdot x = L^{-1} \cdot y$ wordt dan verkregen via achterwaartse substitutie:

```
> xx := BackwardSubstitute( U, MatrixInverse(L).yn ):
  print( 'x' = xx );
```

$$x = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \qquad \textbf{(3.2.2)}$$

We kunnen $L^{-1} \cdot y$ ook eerst uitrekenen door middel van voorwaartse substitutie:

$L^{-1} \cdot y = v$ **then** $L \cdot v = y$

Waarbij we dus van boven naar onder te werk gaan (i.pv. onder naar boven) om de elementen van de vector v één voor één te bepalen door de vorige berekende waardes te subtitueren.

```
> v := ForwardSubstitute(L,yn):
  print( 'v' = v );
```

$$v = \begin{bmatrix} 31 \\ 90 \\ 130 \\ 90 \\ 24 \end{bmatrix} \qquad \textbf{(3.2.3)}$$

```
> print( 'U' = U, 'x' = xn, 'v'=v );
```

$$U = \begin{bmatrix} 1 & 2 & 4 & 8 & 16 \\ 0 & 1 & 5 & 19 & 65 \\ 0 & 0 & 2 & 18 & 110 \\ 0 & 0 & 0 & 6 & 84 \\ 0 & 0 & 0 & 0 & 24 \end{bmatrix}, x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}, v = \begin{bmatrix} 31 \\ 90 \\ 130 \\ 90 \\ 24 \end{bmatrix} \qquad \textbf{(3.2.4)}$$

Implementatie

De volgende procedure lost een stelsel $A \cdot x = y$ op met behulp van **naïeve Gauss-eliminatie** (i.e., zonder pivotering).

```
> GaussianEliminationNoPivoting := proc( A::Matrix, y::Vector
  [column], { verbose := false } )

    local R, n, j;

    R := <A|y>;
    n := RowDimension(R);

    if verbose then print( 'R'[0] = R ) end;
    for j from 1 to n-1 do
      Pivot( R, j, j, [j+1..n], inplace=true );
      if verbose then print(``); print( 'R'[j] = R ) end;
    end;

    BackwardSubstitute( R );

  end proc:

  GENP := GaussianEliminationNoPivoting:
```

We lossen nu het stelsel $\boldsymbol{A \cdot x = y}$ op met naïve Gauss-eliminatie.

```
> print( 'A' = An, 'y' = yn );
```

(3.3.1)

$$A = \begin{bmatrix} 1 & 2 & 4 & 8 & 16 \\ 1 & 3 & 9 & 27 & 81 \\ 1 & 4 & 16 & 64 & 256 \\ 1 & 5 & 25 & 125 & 625 \\ 1 & 6 & 36 & 216 & 1296 \end{bmatrix}, y = \begin{bmatrix} 31 \\ 121 \\ 341 \\ 781 \\ 1555 \end{bmatrix} \quad \textbf{(3.3.1)}$$

```
> 'x' = GENP( An, yn, verbose );
```

$$R_0 = \begin{bmatrix} 1 & 2 & 4 & 8 & 16 & 31 \\ 1 & 3 & 9 & 27 & 81 & 121 \\ 1 & 4 & 16 & 64 & 256 & 341 \\ 1 & 5 & 25 & 125 & 625 & 781 \\ 1 & 6 & 36 & 216 & 1296 & 1555 \end{bmatrix}$$

$$R_1 = \begin{bmatrix} 1 & 2 & 4 & 8 & 16 & 31 \\ 0 & 1 & 5 & 19 & 65 & 90 \\ 0 & 2 & 12 & 56 & 240 & 310 \\ 0 & 3 & 21 & 117 & 609 & 750 \\ 0 & 4 & 32 & 208 & 1280 & 1524 \end{bmatrix}$$

$$R_2 = \begin{bmatrix} 1 & 2 & 4 & 8 & 16 & 31 \\ 0 & 1 & 5 & 19 & 65 & 90 \\ 0 & 0 & 2 & 18 & 110 & 130 \\ 0 & 0 & 6 & 60 & 414 & 480 \\ 0 & 0 & 12 & 132 & 1020 & 1164 \end{bmatrix}$$

$$R_3 = \begin{bmatrix} 1 & 2 & 4 & 8 & 16 & 31 \\ 0 & 1 & 5 & 19 & 65 & 90 \\ 0 & 0 & 2 & 18 & 110 & 130 \\ 0 & 0 & 0 & 6 & 84 & 90 \\ 0 & 0 & 0 & 24 & 360 & 384 \end{bmatrix}$$

$$R_4 = \begin{bmatrix} 1 & 2 & 4 & 8 & 16 & 31 \\ 0 & 1 & 5 & 19 & 65 & 90 \\ 0 & 0 & 2 & 18 & 110 & 130 \\ 0 & 0 & 0 & 6 & 84 & 90 \\ 0 & 0 & 0 & 0 & 24 & 24 \end{bmatrix}$$

$$x = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \qquad \textbf{(3.3.2)}$$

Opmerking: we zouden bovenstaande procedure nog kunnen aanpassen zodat ook L wordt bijgehouden in R. Dit kunnen we doen aangezien we weten dat R een bovendriehoeksstructuur heeft, dus de elementen onder de diagonaal zijn ongebruikt.

2.5. Gauss-eliminatie met partiële pivotering

Implementatie (facultatief)

De volgende procedure lost een stelsel $A \cdot x = y$ op met behulp van **Gauss-eliminatie met partiële pivotering**.

Hierbij maken we gebruik van een index vector l, waarin we de volgorde van de vergelijkingen bijhouden.
We wisselen de rijen dus niet fysiek om, maar we houden gewoon de volgorde bij waarin we werken.

```
> GaussianEliminationPartialPivoting := proc( A::Matrix,
  y::Vector[column], { verbose := false } )

    local R, n, l, j, i;

    R := <A|y>;
    n := RowDimension(A);
    l := [$1..n];

    if verbose then print( 'l'[0] = l ) end;
    if verbose then print( 'R'[0] = R ); print(``) end;
    for j from 1 to n-1 do
      i := max[index]( convert( abs(R[l[j..n],j]), list ) ) +
  j-1;
      # if verbose then print( abs('r'['i',j]) = convert( abs(R
  [l[1..n],j]), list ), 'max' = abs(R[l[i],j]), 'index' = l[i] )
  end;
```

```
    if verbose then print( abs('r'['i',j]) = [seq(_,i=1..(j-1)
), seq(abs(R[l[i],j]),i=j..n)], 'max' = abs(R[l[i],j]),
'index' = l[i] ) end;    l[i], l[j] := l[j], l[i];
    Pivot( R, l[j], j, l[j+1..n], inplace=true );
    R[l[j+1..n],j] := 0;
    if verbose then print( 'l'[j] = l ) end;
    if verbose then print( 'R'[j] = R ); ; print(``) end;

  end;

  # if verbose then print( 'R'[j-1] = R[l[1..n]] ) end;
  BackwardSubstitute( R[l[1..n]] );

end proc:

GEPP := GaussianEliminationPartialPivoting:
```

Oplossen lineair stelsel

We lossen nu het stelsel $A \cdot x = y$ op met Gauss-eliminatie met partiële pivotering.

```
> print( 'A' = An, 'y' = yn );
```

$$A = \begin{bmatrix} 1 & 2 & 4 & 8 & 16 \\ 1 & 3 & 9 & 27 & 81 \\ 1 & 4 & 16 & 64 & 256 \\ 1 & 5 & 25 & 125 & 625 \\ 1 & 6 & 36 & 216 & 1296 \end{bmatrix}, y = \begin{bmatrix} 31 \\ 121 \\ 341 \\ 781 \\ 1555 \end{bmatrix} \qquad \textbf{(4.2.1)}$$

```
> 'x' = GEPP( An, yn, verbose );
```

$$l_0 = [1, 2, 3, 4, 5]$$

$$R_0 = \begin{bmatrix} 1 & 2 & 4 & 8 & 16 & 31 \\ 1 & 3 & 9 & 27 & 81 & 121 \\ 1 & 4 & 16 & 64 & 256 & 341 \\ 1 & 5 & 25 & 125 & 625 & 781 \\ 1 & 6 & 36 & 216 & 1296 & 1555 \end{bmatrix}$$

$$|r_{i,1}| = [1, 1, 1, 1, 1], \max = 1, index = 1$$

$$l_1 = [1, 2, 3, 4, 5]$$

$$R_1 = \begin{bmatrix} 1 & 2 & 4 & 8 & 16 & 31 \\ 0 & 1 & 5 & 19 & 65 & 90 \\ 0 & 2 & 12 & 56 & 240 & 310 \\ 0 & 3 & 21 & 117 & 609 & 750 \\ 0 & 4 & 32 & 208 & 1280 & 1524 \end{bmatrix}$$

$$|r_{i,2}| = [\_, 1, 2, 3, 4], \max = 4, index = 5$$

$$l_2 = [1, 5, 3, 4, 2]$$

$$R_2 = \begin{bmatrix} 1 & 2 & 4 & 8 & 16 & 31 \\ 0 & 0 & -3 & -33 & -255 & -291 \\ 0 & 0 & -4 & -48 & -400 & -452 \\ 0 & 0 & -3 & -39 & -351 & -393 \\ 0 & 4 & 32 & 208 & 1280 & 1524 \end{bmatrix}$$

$$|r_{i,3}| = [\_, \_, 4, 3, 3], \max = 4, index = 3$$

$$l_3 = [1, 5, 3, 4, 2]$$

$$R_3 = \begin{bmatrix} 1 & 2 & 4 & 8 & 16 & 31 \\ 0 & 0 & 0 & 3 & 45 & 48 \\ 0 & 0 & -4 & -48 & -400 & -452 \\ 0 & 0 & 0 & -3 & -51 & -54 \\ 0 & 4 & 32 & 208 & 1280 & 1524 \end{bmatrix}$$

$$|r_{i,4}| = [\_, \_, \_, 3, 3], \max = 3, index = 4$$

$$l_4 = [1, 5, 3, 4, 2]$$

$$R_4 = \begin{bmatrix} 1 & 2 & 4 & 8 & 16 & 31 \\ 0 & 0 & 0 & 0 & -6 & -6 \\ 0 & 0 & -4 & -48 & -400 & -452 \\ 0 & 0 & 0 & -3 & -51 & -54 \\ 0 & 4 & 32 & 208 & 1280 & 1524 \end{bmatrix}$$

$$x = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \qquad \textbf{(4.2.2)}$$

We kunnen de decompositie ook m.b.v. een permutatiematrix P schrijven. Deze permutatiematrix geeft weer in welke volgorde van rijen we de eliminatie hebben toegepast.

```
> print( 'L.U = P.A' );
  P,L,U := Student[NumericalAnalysis][MatrixDecomposition]( An,
  method=PLU ):
  print( 'L' = L, 'U' = U, 'P' = P, 'A' = An );
```

$$L \cdot U = P \cdot A$$

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & \frac{1}{2} & 1 & 0 & 0 \\ 1 & \frac{3}{4} & \frac{3}{4} & 1 & 0 \\ 1 & \frac{1}{4} & \frac{3}{4} & -1 & 1 \end{bmatrix}, U = \begin{bmatrix} 1 & 2 & 4 & 8 & 16 \\ 0 & 4 & 32 & 208 & 1280 \\ 0 & 0 & -4 & -48 & -400 \\ 0 & 0 & 0 & -3 & -51 \\ 0 & 0 & 0 & 0 & -6 \end{bmatrix}, P = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}, \quad \textbf{(4.2.3)}$$

$$A = \begin{bmatrix} 1 & 2 & 4 & 8 & 16 \\ 1 & 3 & 9 & 27 & 81 \\ 1 & 4 & 16 & 64 & 256 \\ 1 & 5 & 25 & 125 & 625 \\ 1 & 6 & 36 & 216 & 1296 \end{bmatrix}$$

Er geldt: $LU = PA$.

```
> Equal( L.U, P.An );
```

$$true \quad \textbf{(4.2.4)}$$

2.8. Resultaten met hardware floats

```
> UseHardwareFloats := true;
```

$$UseHardwareFloats := true \quad \textbf{(5.1)}$$

```
> N := [$5..15];
```

$$N := [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] \quad \textbf{(5.2)}$$

```
> Ahw := table([]):
  yhw := table([]):
  xtilde := table([]):
> for i in N do
    Ahw[i] := HFloat~( A(i) );
    yhw[i] := HFloat~( y(i) ):
    xtilde[i] := GEPP( Ahw[i], yhw[i] ):
    # for j from 1 to 10 do print(``); end:
  end:
> nn := 5:
  print( 'n' = nn );
```

$$n = 5 \quad \textbf{(5.3)}$$

```
> print( 'A'[nn] = Ahw[nn] );
```

$$A_5 = \begin{bmatrix} 1. & 2. & 4. & 8. & 16. \\ 1. & 3. & 9. & 27. & 81. \\ 1. & 4. & 16. & 64. & 256. \\ 1. & 5. & 25. & 125. & 625. \\ 1. & 6. & 36. & 216. & 1296. \end{bmatrix} \quad \textbf{(5.4)}$$

```
> print( 'x'[nn] = xtilde[nn] );
```

$$x_5 = \begin{bmatrix} 1. \\ 1. \\ 1. \\ 1. \\ 1. \end{bmatrix} \quad \textbf{(5.5)}$$

Voor kleine dimensie vinden we de exacte oplossing (in floating-point representatie) terug.

```
> nn := 10:
  print( 'n' = nn );
```

$$n = 10 \quad \textbf{(5.6)}$$

```
> print( 'A'[nn] = Ahw[nn] );
```

$A_{10} = [[1., 2., 4., 8., 16., 32., 64., 128., 256., 512.],$ **(5.7)**

$[1., 3., 9., 27., 81., 243., 729., 2187., 6561., 19683.],$

$[1., 4., 16., 64., 256., 1024., 4096., 16384., 65536., 262144.],$

$[1., 5., 25., 125., 625., 3125., 15625., 78125., 390625., 1.953125\ 10^6],$

$[1., 6., 36., 216., 1296., 7776., 46656., 279936., 1.679616\ 10^6, 1.0077696\ 10^7],$

$[1., 7., 49., 343., 2401., 16807., 117649., 823543., 5.764801\ 10^6, 4.0353607\ 10^7],$

$[1., 8., 64., 512., 4096., 32768., 262144., 2.097152\ 10^6, 1.6777216\ 10^7,$
$1.34217728\ 10^8],$

$[1., 9., 81., 729., 6561., 59049., 531441., 4.782969\ 10^6, 4.3046721\ 10^7,$
$3.87420489\ 10^8],$

$[1., 10., 100., 1000., 10000., 100000., 1.000000\ 10^6, 1.0000000\ 10^7,$
$1.00000000\ 10^8, 1.000000000\ 10^9],$

$[1., 11., 121., 1331., 14641., 161051., 1.771561\ 10^6, 1.9487171\ 10^7,$
$2.14358881\ 10^8, 2.357947691\ 10^9]]$

```
> print( 'x'[nn] = xtilde[nn] );
```

$$x_{10} = \begin{bmatrix} 1.00004398674832 \\ 0.999917968604747 \\ 1.00006472153652 \\ 0.999971531715629 \\ 1.00000772224079 \\ 0.999998655571445 \\ 1.00000015071885 \\ 0.999999989477472 \\ 1.00000000041624 \\ 0.999999999992876 \end{bmatrix} \quad \textbf{(5.8)}$$

We merken op dat de oplossing al iets minder nauwkeurig is na het verhogen van de dimensie van het stelsel.

```
> nn := 15:
  print( 'n' = nn );
```

$$n = 15 \quad \textbf{(5.9)}$$

```
> print( 'A'[nn] = Ahw[nn] );
```

$A_{15} = [[1., 2., 4., 8., 16., 32., 64., 128., 256., 512., 1024., 2048., 4096., 8192., 16384.$ **(5.10)**

$],$

$[1., 3., 9., 27., 81., 243., 729., 2187., 6561., 19683., 59049., 177147., 531441.,$
$1.594323\ 10^{6}, 4.782969\ 10^{6}],$

$[1., 4., 16., 64., 256., 1024., 4096., 16384., 65536., 262144., 1.048576\ 10^{6},$
$4.194304\ 10^{6}, 1.6777216\ 10^{7}, 6.7108864\ 10^{7}, 2.68435456\ 10^{8}],$

$[1., 5., 25., 125., 625., 3125., 15625., 78125., 390625., 1.953125\ 10^{6},$
$9.765625\ 10^{6}, 4.8828125\ 10^{7}, 2.44140625\ 10^{8}, 1.220703125\ 10^{9}, 6.103515625\ 10^{9}],$

$[1., 6., 36., 216., 1296., 7776., 46656., 279936., 1.679616\ 10^{6}, 1.0077696\ 10^{7},$
$6.0466176\ 10^{7}, 3.62797056\ 10^{8}, 2.176782336\ 10^{9}, 1.3060694016\ 10^{10},$
$7.8364164096\ 10^{10}],$

$[1., 7., 49., 343., 2401., 16807., 117649., 823543., 5.764801\ 10^{6}, 4.0353607\ 10^{7},$
$2.82475249\ 10^{8}, 1.977326743\ 10^{9}, 1.3841287201\ 10^{10}, 9.6889010407\ 10^{10},$
$6.78223072849\ 10^{11}],$

$[1., 8., 64., 512., 4096., 32768., 262144., 2.097152\ 10^{6}, 1.6777216\ 10^{7},$
$1.34217728\ 10^{8}, 1.073741824\ 10^{9}, 8.589934592\ 10^{9}, 6.8719476736\ 10^{10},$
$5.49755813888\ 10^{11}, 4.398046511104\ 10^{12}],$

[1., 9., 81., 729., 6561., 59049., 531441., $4.782969\ 10^{6}$, $4.3046721\ 10^{7}$,
$3.87420489\ 10^{8}$, $3.486784401\ 10^{9}$, $3.1381059609\ 10^{10}$, $2.82429536481\ 10^{11}$,
$2.541865828329\ 10^{12}$, $2.2876792454961\ 10^{13}$],
[1., 10., 100., 1000., 10000., 100000., $1.000000\ 10^{6}$, $1.0000000\ 10^{7}$,
$1.00000000\ 10^{8}$, $1.000000000\ 10^{9}$, $1.0000000000\ 10^{10}$, $1.00000000000\ 10^{11}$,
$1.000000000000\ 10^{12}$, $1.0000000000000\ 10^{13}$, $1.00000000000000\ 10^{14}$],
[1., 11., 121., 1331., 14641., 161051., $1.771561\ 10^{6}$, $1.9487171\ 10^{7}$,
$2.14358881\ 10^{8}$, $2.357947691\ 10^{9}$, $2.5937424601\ 10^{10}$, $2.85311670611\ 10^{11}$,
$3.138428376721\ 10^{12}$, $3.4522712143931\ 10^{13}$, $3.79749833583241\ 10^{14}$],
[1., 12., 144., 1728., 20736., 248832., $2.985984\ 10^{6}$, $3.5831808\ 10^{7}$,
$4.29981696\ 10^{8}$, $5.159780352\ 10^{9}$, $6.1917364224\ 10^{10}$, $7.43008370688\ 10^{11}$,
$8.916100448256\ 10^{12}$, $1.06993205379072\ 10^{14}$, $1.28391846454886\ 10^{15}$],
[1., 13., 169., 2197., 28561., 371293., $4.826809\ 10^{6}$, $6.2748517\ 10^{7}$,
$8.15730721\ 10^{8}$, $1.0604499373\ 10^{10}$, $1.37858491849\ 10^{11}$, $1.792160394037\ 10^{12}$,
$2.3298085122481\ 10^{13}$, $3.02875106592253\ 10^{14}$, $3.93737638569929\ 10^{15}$],
[1., 14., 196., 2744., 38416., 537824., $7.529536\ 10^{6}$, $1.05413504\ 10^{8}$,
$1.475789056\ 10^{9}$, $2.0661046784\ 10^{10}$, $2.89254654976\ 10^{11}$, $4.049565169664\ 10^{12}$,
$5.6693912375296\ 10^{13}$, $7.93714773254144\ 10^{14}$, $1.11120068255580\ 10^{16}$],
[1., 15., 225., 3375., 50625., 759375., $1.1390625\ 10^{7}$, $1.70859375\ 10^{8}$,
$2.562890625\ 10^{9}$, $3.8443359375\ 10^{10}$, $5.76650390625\ 10^{11}$, $8.649755859375\ 10^{12}$,
$1.29746337890625\ 10^{14}$, $1.94619506835938\ 10^{15}$, $2.91929260253906\ 10^{16}$],
[1., 16., 256., 4096., 65536., $1.048576\ 10^{6}$, $1.6777216\ 10^{7}$, $2.68435456\ 10^{8}$,
$4.294967296\ 10^{9}$, $6.8719476736\ 10^{10}$, $1.099511627776\ 10^{12}$, $1.7592186044416\ 10^{13}$,
$2.81474976710656\ 10^{14}$, $4.50359962737050\ 10^{15}$, $7.20575940379279\ 10^{16}$]]

```
> print( 'x'[nn] = xtilde[nn] );
```

(5.11)

$$x_{15} = \begin{bmatrix} -19457.5502911755 \\ 43516.5171453505 \\ -43189.9565513271 \\ 25321.3638101348 \\ -9828.83342597758 \\ 2682.86041129300 \\ -530.819434162558 \\ 79.0756800431057 \\ -7.54764817090229 \\ 1.69595368938154 \\ 0.958435965387637 \\ 1.00176863641223 \\ 0.999949234487738 \\ 1.00000088093507 \\ 0.999999993019572 \end{bmatrix} \quad \textbf{(5.11)}$$

In het geval $n = 15$ zien we dat de oplossing ver verwijderd is van de eigenlijke oplossing van het stelsel. Hoe kunnen we dit verklaren?

2.8. en 2.9. Conditionering en stabiliteit

Conditiegetal

Het conditiegetal van een matrix wordt gedefinieerd door

$$\kappa_p(A) = \frac{\max\left(\frac{\|Ax\|_p}{\|x\|_p}\right)}{\min\left(\frac{\|Ax\|_p}{\|x\|_p}\right)} = \|A\|_p \, \|A^{-1}\|_p .$$

Hierbij neemt men vaak $p = \infty$, met $\|A\| = \|A\|_\infty = \max \sum_{j=1}^{n} |a_{ij}|$, of $p = 2$.

Het conditiegetal geeft a priori informatie over de kwaliteit van de output van een algoritme. In het ideale geval is $\kappa_p(A) = 1$. Dit wil zeggen dat de fouten op de input niet doorwerken en vergroten.

```
> unassign('n'):
  condh := Matrix( nops(N)+1, 2 ):
  condh[1] := < 'n', 'cond(A)' >:
```

```
for i from 1 to nops(N) do
  condh[i+1] := < N[i], evalf[5]( ConditionNumber( Ahw[N[i]] )
) >;
end:
print( condh );
```

n	$cond(A)$
5	232210.
6	$8.3872\ 10^{6}$
7	$3.4194\ 10^{8}$
8	$1.5607\ 10^{10}$
9	$7.9051\ 10^{11}$
10	$4.4070\ 10^{13}$
11	$2.6845\ 10^{15}$
12	$1.7751\ 10^{17}$
13	$1.2670\ 10^{19}$
14	$9.7116\ 10^{20}$
15	$7.9821\ 10^{22}$

(6.1.1)

We stellen vast dat het conditiegetal zeer snel stijgt in functie van de dimensie van de matrix (i.e., met n). Dit stelsel is dus slecht geconditioneerd voor grote dimensies.
Dit verklaart natuurlijk waarom de fout voor $n = 15$ heel groot was.

Residu vector

Als $\tilde{x}$ de berekende oplossing is voor het stelsel $A \cdot x = y$, dan wordt de residu vector gegeven door $r = y - A\,\tilde{x}$.

Het residu geeft eigenlijk weer hoe goed onze berekende oplossing aan de vergelijkingen van het stelsel voldoet.
Verwar dit zeker niet met de error of fout!

```
> r := table([]):
  for i from 1 to nops(N) do
    r[N[i]] := yhw[N[i]] - Ahw[N[i]] . xtilde[N[i]]:
  end:
```

Voor Gaussische eliminatie met partiële pivotering zou het relatieve residu van de grootte-orde van de machine epsilon ϵ moeten zijn.

Met andere woorden zullen we dus controleren dat $\frac{\| y - A\,\tilde{x} \|}{\| A \| \, \| \tilde{x} \|} \leq \rho\epsilon$.

```
> check1 := Matrix( nops(N)+1, 5 ):
  xt := `#mover(mi("x"),mo("&sim;"))`:
  check1[1] := < 'n', ''norm''(r), ''norm''(A), ''norm''(xt),
  ''norm''(r)/(''norm''(A)*''norm''(xt)) >:
  for i from 1 to nops(N) do
    Nr := Norm( r[N[i]] ):
    NAhw := Norm( Ahw[N[i]] ):
    Nxtilde := Norm( xtilde[N[i]] ):
    check1[i+1] := < N[i], evalf[5]( Nr ), evalf[5]( NAhw ),
  evalf[5]( Nxtilde ), evalf[5]( Nr/(NAhw*Nxtilde) ) >;
  end:
  print( check1 );
```

$$\begin{bmatrix} n & 'norm'(r) & 'norm'(A) & 'norm'(\tilde{x}) & \frac{'norm'(r)}{'norm'(A)\ 'norm'(\tilde{x})} \\ 5 & 0. & 1555. & 1. & 0. \\ 6 & 0. & 19608. & 1. & 0. \\ 7 & 0. & 299590. & 1. & 0. \\ 8 & 0. & 5.3808\ 10^{6} & 1. & 0. \\ 9 & 0. & 1.1111\ 10^{8} & 1. & 0. \\ 10 & 1.8626\ 10^{-8} & 2.5937\ 10^{9} & 1.0001 & 7.1808\ 10^{-18} \\ 11 & 9.5367\ 10^{-7} & 6.7546\ 10^{10} & 1.0008 & 1.4107\ 10^{-17} \\ 12 & 0.000015259 & 1.9415\ 10^{12} & 1.0647 & 7.3818\ 10^{-18} \\ 13 & 0.0042114 & 6.1055\ 10^{13} & 139.41 & 4.9479\ 10^{-19} \\ 14 & 0.062392 & 2.0852\ 10^{15} & 1253.8 & 2.3864\ 10^{-20} \\ 15 & 8. & 7.6861\ 10^{16} & 43517. & 2.3918\ 10^{-21} \end{bmatrix} \quad \textbf{(6.2.1)}$$

```
> eps := evalf[5]( 2^(-52) );
```

$$eps := 2.2204\ 10^{-16} \quad \textbf{(6.2.2)}$$

Error vector

Als $\tilde{x}$ een berekende oplossing is voor het stelsel $A \cdot x = y$ en x^* de exacte oplossing, dan wordt de error vector gegeven door $e = x^* - \tilde{x}$.

De error of fout geeft dus weer hoe ver we verwijderd zijn van de echte (exacte) oplossing van het stelsel lineaire vergelijkingen.
Verwar dit zeker niet met het residu!

Als we de exacte oplossing niet kennen, kunnen we de relatieve error alleen afschatten :

$$\frac{\|x^* - \tilde{x}\|}{\|x^*\|} \leq \rho\epsilon\kappa(A) .$$

Dit geeft ons a priori een beeld van het worst-case scenario.

```
> check2 := Matrix( nops(N)+1, 2 ):
  check2[1] := < 'n', 'epsilon . kappa (A)' >:
> for i from 1 to nops(N) do
    condA := ConditionNumber( Ahw[N[i]] ):
    check2[i+1] := < N[i], evalf[5]( eps*condA ) >;
  end:
  print( check2 );
```

$$\begin{bmatrix} n & \epsilon \cdot \kappa(A) \\ 5 & 5.1560\ 10^{-11} \\ 6 & 1.8622\ 10^{-9} \\ 7 & 7.5924\ 10^{-8} \\ 8 & 3.4654\ 10^{-6} \\ 9 & 0.00017552 \\ 10 & 0.0097851 \\ 11 & 0.59607 \\ 12 & 39.414 \\ 13 & 2813.2 \\ 14 & 215640. \\ 15 & 1.7724\ 10^{7} \end{bmatrix} \quad \textbf{(6.3.1)}$$

Vermits we hier wel de exacte oplossing x^* kennen, kunnen we de relatieve error ook echt berekenen.

```
> check3 := Matrix( nops(N)+1, 4 ):
  check3[1] := < 'n', 'norm (e)', 'norm (e)' / 'norm (x)',
  'epsilon . kappa (A)' >:
  for i from 1 to nops(N) do
    abserror :=  Norm( xtilde[N[i]] - Vector(N[i],1) );
    Nx := Norm( Vector(N[i],1) );
    relerror := abserror/Nx;
    check3[i+1] := < N[i], evalf[5]( abserror ), evalf[5](
  relerror),  evalf[5](eps*ConditionNumber( Ahw[N[i]] ))>;
  end:
  print( check3 );
```

(6.3.2)

n	$norm(e)$	$\frac{norm(e)}{norm(x)}$	$\epsilon \cdot \kappa(A)$
5	0.	0.	$5.1560\ 10^{-11}$
6	0.	0.	$1.8623\ 10^{-9}$
7	0.	0.	$7.5924\ 10^{-8}$
8	0.	0.	$3.4654\ 10^{-6}$
9	0.	0.	0.00017552
10	0.000082031	0.000082031	0.0097853
11	0.00080670	0.00080670	0.59607
12	0.075316	0.075316	39.414
13	138.41	138.41	2813.2
14	1252.8	1252.8	215640.
15	43516.	43516.	$1.7723\ 10^{7}$

(6.3.2)