Editor - Z:\University\Numerik1\Ue3\bsp2.m

gauss.m | gauss_pivot_scaled.m | gauss_pivot.m | r.m | bsp2.m

```
1  A = [3.03,-12.1,14;-3.03,12.1,-7;6.11,-14.2,21];
2  b = [-119;120;-139];
3
4  disp('a')
5  disp(gauss(A,b,1));
6  disp('b')
7  disp(gauss_pivot(A,b,1));
8
9  disp('c')
10 disp(gauss_pivot(A,b,2));
11
12 disp('d')
13 disp(gauss_pivot_scaled(A,b,3));
14
15
```

Command Window

```
>> bsp2
a
   NaN   NaN   NaN

b
     0    10     0

c
     0    10     0

d
          0   10.0000    0.1430

>>
```

Editor - Z:\University\Numerik1\Ue3\r.m

gauss.m | gauss_pivot_scaled.m | gauss_pivot.m | r.m

```
1  function [ n ] = r( n, prec )
2  %ROUND Summary of this function goes here
3  %   Detailed explanation goes here
4
5
6      n = round(n, prec, 'significant');
7  end
8
9
```

Editor - Z:\University\Numerik1\Ue3\gauss.m

+7 | gauss.m | gauss_backwardSubstitution.m | gauss_pivot.m | gauss_eliminate_pivot.m | gauss_eliminate_base.m | +

```
1  function [ x ] = gauss_eliminate( A, b, prec )
2  %GAUSS_ELIMINATE Summary of this function goes here
3  %   Detailed explanation goes here
4  % if A is a regular square n×n-matrix with no 0's on the diagonal
5  % and b is a 1×n vector
6  % then A and b will be converted to "standard form"
7      A = [A,b];
8      n = min(size(A));
9
10     % start with the matrix rounded correctly
11     for i=1:size(A,1)
12         for j = 1:size(A,2)
13             A(i,j) = r(A(i,j), prec);
14         end
15     end
16     %disp(A)
17
18     for k = 1:n-1
19         for i = k+1:n
20             A(i,k) = r(A(i,k) / A(k,k), prec);
21
22             for j = k+1:n+1
23                 A(i,j) = r(A(i,j) - r(A(i,k) * A(k,j), prec), prec);
24             end
25         end
26         %disp(A);
27     end
28
29     x(n) = r(A(n,n+1) / A(n,n), prec);
30     for i = n-1:-1:1
31         s = 0;
32         for j = i+1:n
33             s = r(s + r(A(i,j) * x(j), prec), prec);
34         end
35         x(i) = r(r(A(i, n+1) - s, prec) / A(i,i), prec);
36     end
37 end
38
39
```

Editor - Z:\University\Numerik1\Ue3\gauss_pivot.m

gauss.m | gauss_pivot_scaled.m | gauss_pivot.m

```
function [ x ] = gauss_pivot( A, b, prec )
%GAUSS_PIVOT Summary of this function goes here
%   Detailed explanation goes here
    A = [A,b];
    n = min(size(A));

    % start with the matrix rounded correctly
    for i=1:size(A,1)
        for j = 1:size(A,2)
            A(i,j) = r(A(i,j), prec);
        end
    end
    I = 1:n;

    for k = 1:n-1
        [tmp, p] = max(abs(A(I(k:n), k)));
        p = p + k - 1; % adjust offset

        % regulär? stop?
        It = I(k); I(k) = I(p);  I(p) = It;
        for i = k+1:n
            A(I(i), k) = r(A(I(i),k) / A(I(k), k), prec);
            for j = k+1:n+1
                A(I(i), j) = r(A(I(i),j) - r(A(I(i),k) * A(I(k),j),prec),prec);
            end
        end
        %disp(A);
    end

    x(n) = r(A(I(n),n+1) / A(I(n),n), prec);
    for i = n-1:-1:1
        s = 0;
        for j = i+1:n
            s = r(s + r(A(I(i),j) * x(j), prec), prec);
        end
        x(i) = r(r(A(I(i), n+1) - s, prec) / A(I(i),i), prec);
    end
end
```

Editor - Z:\University\Numerik1\Ue3\gauss_pivot_scaled.m*

gauss.m | gauss_pivot_scaled.m* | gauss_pivot.m

```
function [ x ] = gauss_pivot_scaled( A, b, prec )
%GAUSS_PIVOT_SCALED Summary of this function goes here
    A = [A,b];
    n = min(size(A));
    s = max(abs(A'));
    I = 1:n;

    % start with the matrix rounded correctly
    for i=1:size(A,1)
        for j = 1:size(A,2)
            A(i,j) = r(A(i,j), prec);
        end
    end
    %disp(A)

    for k = 1:n-1
        [tmp, p] = max(abs(A(I(k:n), k)' ./ s(I(k:n))));
        p = p + k - 1; % adjust offset

        % regulär? stop?
        It = I(k); I(k) = I(p);  I(p) = It;
        for i = k+1:n
            A(I(i), k) = r(A(I(i),k) / A(I(k), k), prec);
            for j = k+1:n+1
                A(I(i), j) = r(A(I(i),j) - r(A(I(i),k) * A(I(k),j),prec),prec);
            end
        end
        % disp(A);
    end

    x(n) = r(A(I(n),n+1) / A(I(n),n), prec);
    for i = n-1:-1:1
        s = 0;
        for j = i+1:n
            s = r(s + r(A(I(i),j) * x(j), prec), prec);
        end
        x(i) = r(r(A(I(i), n+1) - s, prec) / A(I(i),i), prec);
    end
end
```

Aufgabe 4

Für eine gegebene Matrix wollen wir verschiedene Varianten der Gauß-Elimination implementieren, um eine LU-Zerlegung zu erhalten. Dazu betrachten wir erstmals in a.) und b.) die Gauß-Elimination ohne Pivotsuche.

Die Matrix ist gegeben als

```
A = spdiags(ones(n,3),[-p,0,q],n,n) + ep * speye(n);
```

für $p, q \in \mathbb{N}$ und $\epsilon > 1$.

Code der Gauß-Elemination ohne Pivotsuche:

```
A = spdiags(ones(n,3),[-p,0,q],n,n) + ep * speye(n);

B = A;
for k = 1:(n-1)
  for i = (k+1):n
    A(i,k) = A(i,k) / A(k,k);
    for j = (k+1):n
      A(i,j) = A(i,j) - A(i,k) * A(k,j);

    endfor
  endfor
endfor

%cut the triangular matrices out of A
L = tril(A,-1) + eye(n);
U = triu(A);

%0 if correct factorisation
full(sum(abs(L*U-B)(:)))
```

a.) richtig
Beispielhafte Konfiguration:

```
n = 6;
p = 3;
q = 4;
ep = 1.5;
```

Resultat:

L	U

`>> full(L)`
`ans =`
1.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 1.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 1.00000 0.00000 0.00000 0.00000
0.40000 0.00000 0.00000 1.00000 0.00000 0.00000
0.00000 0.40000 0.00000 0.00000 1.00000 0.00000
0.00000 0.00000 0.40000 0.00000 0.00000 1.00000
	`>> full(U)`
	`ans =`
	2.50000 0.00000 0.00000 0.00000 1.00000 0.00000
	0.00000 2.50000 0.00000 0.00000 0.00000 1.00000
	0.00000 0.00000 2.50000 0.00000 0.00000 0.00000
	0.00000 0.00000 0.00000 2.50000 -0.40000 0.00000
	0.00000 0.00000 0.00000 0.00000 2.50000 -0.40000
	0.00000 0.00000 0.00000 0.00000 0.00000 2.50000

Wir sehen hier, dass die nicht-trivialen Elemente zwischen den Diagonalen $-p$ bis 0 bzw 0 bis q.

b.) richtig

Beispielhafte Konfiguration:

```
n = 6;
p = 3;
q = 3;
ep = 1.2;
```

Resultat:

L	U

```
>> full(L)
ans =

   1.00000   0.00000   0.00000   0.00000   0.00000   0.00000
   0.00000   1.00000   0.00000   0.00000   0.00000   0.00000
   0.00000   0.00000   1.00000   0.00000   0.00000   0.00000
   0.45455   0.00000   0.00000   1.00000   0.00000   0.00000
   0.00000   0.45455   0.00000   0.00000   1.00000   0.00000
   0.00000   0.00000   0.45455   0.00000   0.00000   1.00000
```

```
>> full(U)
ans =

   2.20000   0.00000   0.00000   1.00000   0.00000   0.00000
   0.00000   2.20000   0.00000   0.00000   1.00000   0.00000
   0.00000   0.00000   2.20000   0.00000   0.00000   1.00000
   0.00000   0.00000   0.00000   1.74545   0.00000   0.00000
   0.00000   0.00000   0.00000   0.00000   1.74545   0.00000
   0.00000   0.00000   0.00000   0.00000   0.00000   1.74545
```

Für Aufgabe c.) benötigen wir die Gauß-Elimination mit Pivotsuche (in den Zeilen)

Code:

```
%Initialize virtual row switch
I=1:n;
for k = 1:(n-1)
  %find pivot element
  [m, p] = max(abs(A(I(k:end),k)));
  %index shift
  p = p + k - 1;
  %abandon to avoid dividing by 0 if A is irregulär
  if A(I(p),k) == 0
    break;
  endif
  %virtual row switch
  it = I(k); I(k) = I(p); I(p) = it;

  for i = (k+1):n
    A(I(i),k) = A(I(i),k) / A(I(k),k);
    for j = (k+1):n
      A(I(i),j) = A(I(i),j) - A(I(i),k) * A(I(k),j);
    endfor
  endfor
endfor
```

```
%create permutation matrix
P = zeros(n,n);
for i = 1:n
  for j = 1:n
    P(i,j) = (I(i) == j);
  endfor
endfor

%cut the triangular matrices out of A
L = tril(A(I,:),-1) + eye(n);
U = triu(A(I,:));

%0 if correct factorisation
full(sum(abs(L*U-P*B)(:)))
```

c.) überarbeitet

Für Aufgabe c.) lassen sich in der ursprünglichen Aufgabenstellung $\epsilon > 0$ Gegenbeispiele finden. Wählen wir jedoch $\epsilon = \sqrt{2} - 1$ werden Pivot-Elemente 0 und der Algorithmus funktioniert nicht mehr. Daher die Bedingung $\epsilon > 1$.
Nun ist die Aussage korrekt.

Beispielhafte Konfiguration

```
n = 8;
p = 3;
q = 3;
ep = 1.2;
```

Resultat:

L	U
>> full(L) ans = 1.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 1.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 1.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.45455 0.00000 0.00000 1.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.45455 0.00000 0.00000 1.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.45455 0.00000 0.00000 1.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.57292 0.00000 0.00000 1.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.57292 0.00000 0.00000 1.00000	>> full(U) ans = 2.20000 0.00000 0.00000 1.00000 0.00000 0.00000 0.00000 0.00000 0.00000 2.20000 0.00000 0.00000 1.00000 0.00000 0.00000 0.00000 0.00000 0.00000 2.20000 0.00000 0.00000 1.00000 0.00000 0.00000 0.00000 0.00000 0.00000 1.74545 0.00000 0.00000 1.00000 0.00000 0.00000 0.00000 0.00000 0.00000 1.74545 0.00000 0.00000 1.00000 0.00000 0.00000 0.00000 0.00000 0.00000 1.74545 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 1.62708 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 1.62708

Für Aufgabe d.) benötigen wir die Gauß-Elimination mit totaler Pivotsuche.
Code:

```
%init I,J
I = 1:n;
J = 1:n;
for k = 1:(n-1)
  %find pivot element
  size = n - k + 1;
  [m, p] = max(abs(A(I(k:end),J(k:end)).'(:)));
  %calculate correct index
  p_i = ceil(p / size);
  p_j = mod(p, size);
  p_i = p_i + k - 1;
  p_j = p_j + k -1 ;
  %abandon to avoid div by 0 if matrix is irregular
  if B(I(p_i),J(p_j)) == 0
    break;
  endif
  %swap iterator
  it = I(k);
  I(k) = I(p_i);
  I(p_i) = it;
  it = J(k);
  J(k) = J(p_j);
  J(p_j) = it;
```

```
  for i = (k+1):n
    A(I(i),J(k)) = A(I(i),J(k)) / A(I(k),J(k));
    for j = (k+1):(n)
      A(I(i),J(j)) = A(I(i),J(j)) - A(I(i),J(k)) * A(I(k),J(j));
    endfor
  endfor
endfor

%calculate permutation matrices
P = zeros(n,n);
for i = 1:n
  for j = 1:n
    P(i,j) = (I(i) == j);
  endfor
endfor
Q = zeros(n,n);
for i = 1:n
  for j = 1:n
    Q(i,j) = (J(j) == i);
  endfor
endfor
%cut the triangular matrices out of A
L = tril(A(I,J),-1) + eye(n);
U = triu(A(I,J));

%0 if correct factorisation
full(sum(abs(L*U-P*B*Q)(:)))
```

d.) falsch

Wir finden hier ein Gegenbeispiel mit der Konfiguration

```
n = 8;
p = 4;
q = 4;
ep = 2;
```

Resultat für $L + L^T$

```
 2.00000   0.00000   0.00000   0.00000   0.33333   0.00000   0.00000   0.00000
 0.00000   2.00000   0.00000   0.00000   0.00000   0.33333   0.00000   0.00000
 0.00000   0.00000   2.00000   0.00000   0.00000   0.00000   0.33333   0.00000
 0.00000   0.00000   0.00000   2.00000   0.00000   0.00000   0.00000   0.33333
 0.33333   0.00000   0.00000   0.00000   2.00000   0.00000   0.00000   0.00000
 0.00000   0.33333   0.00000   0.00000   0.00000   2.00000   0.00000   0.00000
 0.00000   0.00000   0.33333   0.00000   0.00000   0.00000   2.00000   0.00000
 0.00000   0.00000   0.00000   0.33333   0.00000   0.00000   0.00000   2.00000
```

Diese Matrix hat aber Bandbreite $p + q + 1 = 4 + 4 + 1 = 9 \neq 3$.

Beispiel 6

Ang. $\omega = \text{sqrt}(x) \Rightarrow$

```
w = x;
for i = 1:10
    w = (w + x/w)/2;
end
```

Anzahl der flops der Cholesky-Zerlegung a) ohne Wurzel-Rechnung
b) mit Wurzel-Iteration

a) ① for $i = 2,\dots,n$
$L_{i1} = A_{i1} / L_{11}$ } Division: $\sum_{i=1}^{n-1} 1 = \underline{n-1}$

② for $j = 2,\dots,n-1$
for $k = 1,\dots,j-1$
$s = s + L_{jk} * L_{jk}$ } Add: $\sum_{j=1}^{n-2} j = \frac{(n-2)(n-1)}{2}$ Mult: $-\!\!-$ } $2\sum_{j=1}^{n-2} j = 2\frac{(n-2)(n-1)}{2} = \underline{(n-2)(n-1)}$

③ for $j = 2,\dots,n-1$
for $i = j+1,\dots,n$
for $k = 1,\dots,j-1$
$s = s + L_{ik} * L_{jk}$ } Add: $\sum_{j=1}^{n-2} (n-j-1)\,j$ Mult: $-\!\!-$ } $2\sum_{j=1}^{n-2} (n-j-1)\,j$

$= 2\left(n\sum_{j=1}^{n-2} j - \sum_{j=1}^{n-2} j^2 - \sum_{j=1}^{n-2} j\right)$

$= 2\left(n\frac{(n-2)(n-1)}{2} - \frac{(n-2)(n-1)(2(n-2)+1)}{6} - \frac{(n-2)(n-1)}{2}\right)$

$= \underline{\frac{1}{3}(n-2)(n-1)n}$

④ for $j = 2,\dots,n-1$
for $i = j+1,\dots,n$
$L_{ij} = [A_{ij} - s] / L_{ji}$

Sub: $\sum_{j=1}^{n-2} (n-j-1)$
Dir: $-\!\!-$ } $2\sum_{j=1}^{n-2} (n-j-1) = 2\cdot\left(n\sum_{j=1}^{n-2} 1 - \sum_{j=1}^{n-2} j - \sum_{j=1}^{n-2} 1\right)$

$= 2\left(n(n-2) - \frac{(n-2)(n-1)}{2} - (n-2)\right)$

$= \underline{(n-2)(n-1)}$

⑤ for $k = 1, \dots, n-1$
$s = s + L_{nk} * L_{nk}$
} Add: $\sum_{k=1}^{n-1} 1 = (n-1) \cdot 2$
Mult: -"-

⑥ for $j = 2, \dots, n-1$
$L_{jj} = \sqrt{A_{jj} - s}$
} Sub: $\sum_{j=1}^{n-2} 1 = \sum_{j=1}^{n-1} 1 - 1 = (n-1) - 1$
} $(n-1)$

⑦ $L_{nn} = \sqrt{A_{nn} - s}$ } Sub: 1

Flops: ① + ② + ③ + ④ + ⑤ + ⑥ + ⑦

$= \underbrace{4 \cdot (n-1)}_{①⑤⑥⑦} + \underbrace{2(n-1)(n-2)}_{②④} + \underbrace{\frac{1}{3}(n-2)(n-1)n}_{③} = \frac{1}{3}n(n^2+3n-4)$

= Ergebnis c)

b) mit Wurzel-Iteration

for $i = 1:10$
$\omega = (\omega + x/\omega)/2;$
} Div: $2 \cdot 10 = 20$
Add: 10
} 30

Zusätzlich zu a) ⑧ $j = 2, \dots, n-1$
$L_{jj} = \sqrt{A_{jj} - s}$
} Wurzel: $\sum_{j=1}^{n-2} 30 = 30 \sum_{j=1}^{n-2} 1$
$= 30(n-2)$

⑨ $L_{11} = \sqrt{A_{11}}$
$L_{nn} = \sqrt{A_{nn} - s}$
} Wurzel: $2 \cdot 30 = 60$

Flops: $\underbrace{①-⑦}_{\frac{1}{3}n(n^2+3n-4)} + ⑧ + ⑨ = \frac{1}{3}n(n^2+3n-4) + 30(n-2) + 60$

$= \frac{1}{3}n(n^2+3n+86)$

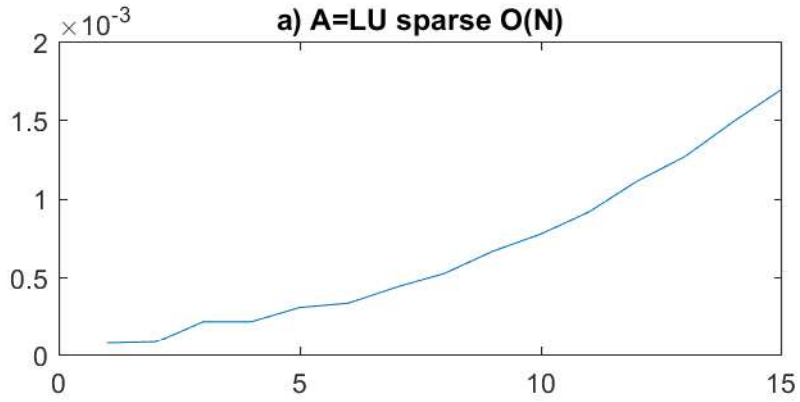
= Ergebnis a)

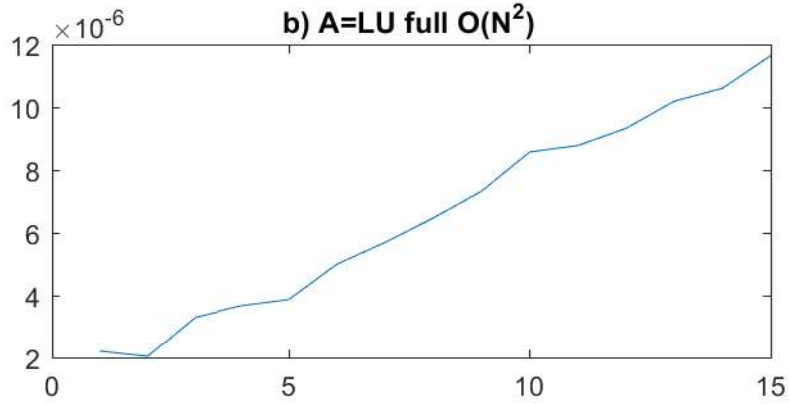
$\Rightarrow$ a), c) korrekt!

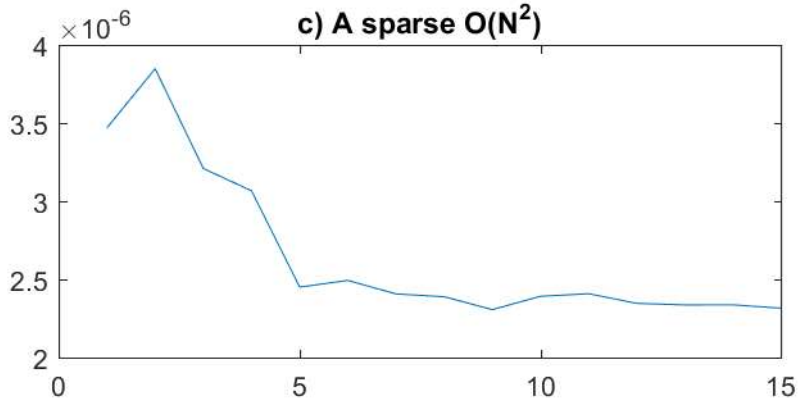
```
% Beispiel 8 Numerik 1 für 23.10.2020
clear
N = 400;
T = 100;
v = 0.4;
timelimita = [];
timelimitb = [];
timelimitc = [];
timelimitd = [];

% beginnen nicht für ganz kleine N, da uns das Verhalten für größer
% werdende N interessiert
for i = 50:25:N;
h = 1/i;
tau = T/i;
d = [-ones(i-1,1),ones(i-1,1)];
Q = spdiags(d,[0,+1],i-1,i);
B = v/h*[sparse(i-1,i-1),Q;-transpose(Q),sparse(i,i)]; %Dimension 2*i-1
A = speye(2*i-1)-(tau/2)*B;
C = speye(2*i-1)+(tau/2)*B;
Cs = sparse(C);
Cf = full(C);
    for k = 1:i;
        x(k,1) = (k-1/2)*h;
    end
a0 = cos(pi*(2*x-1));
W0 = [(v/h)*(Q*a0); zeros(i,1)];

% es gilt die Iteration A * W^k = C * W^(k-1)
% a
Wa = W0;
[L,U] = lu(A);
L = sparse(L);  % eigentlich nicht notwendig, da bereits im sparse format
U = sparse(U);
tic
    for k = 2:i+1;  % wir haben Wa_0 aber Index beginnt mit 1
        Wa(:,k) = U\(L\(Cs*Wa(:,k-1)));
    end
timea = toc;
timelimita = [timelimita, timea/i];% dividieren gleich durch "N" bzw i,
% da uns der Grenzwert interessiert (Definition von big O)

% b
Wb = W0;
Ab = full(A);
Lb = full(L);
Ub = full(U);
tic
    for k = 2:i+1;  % wir haben Wb_0 aber Index beginnt mit 1
        Wb(:,k) = Ub\(Lb\(Cf*Wb(:,k-1)));
    end
timeb = toc;
timelimitb = [timelimitb, timeb/i^2];

% c
Wc = W0;
Ac = sparse(A);
tic
    for k = 2:i+1;  % wir haben Wc_0 aber Index beginnt mit 1
        Wc(:,k) = Ac\(Cs*Wc(:,k-1));
```

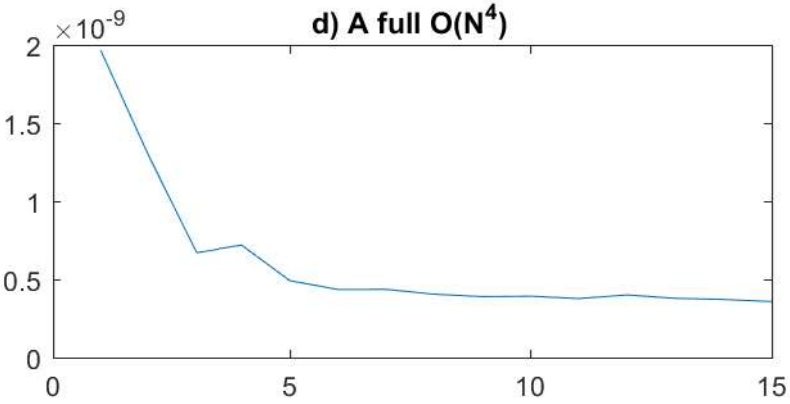
```
    end
timec = toc;
timelimitc = [timelimitc, timec/i^2];

% d
Wd = W0;
Ad = full(A);
tic
    for k  =2:i+1;  % wir haben Wd_0 aber Index beginnt mit 1
        Wd(:,k) = Ad\(Cf*Wd(:,k-1));
    end
timed = toc;
timelimitd = [timelimitd, timed/i^4];

end

% zur Kontrolle - Auslenkung einer Saite
h = 1/N;
tau = T/N;
d = [-ones(N-1,1),ones(N-1,1)];
Q = spdiags(d,[0,+1],N-1,N);
B = v/h*[sparse(N-1,N-1),Q;-transpose(Q),sparse(N,N)]; %Dimension 2*N-1
A = speye(2*N-1)-(tau/2)*B;
C = speye(2*N-1)+(tau/2)*B;
As = sparse(A);
Cs = sparse(C);
% Variablen nochmals belegt, da nur für ""N"" betrachtet

ws = W0;
a = a0;
for k = 2:N+1;
    ws(:,k) = As\(Cs*ws(:,k-1));
    a(:,k) = a(:,k-1)+(tau/2)*(ws(N:2*N-1,k)+ws(N:2*N-1,k-1));
end

hold on
subplot(3,2,1)
plot(timelimita)
title('a) A=LU sparse O(N)')

subplot(3,2,2)
plot(timelimitb)
title('b) A=LU full O(N^2)')

subplot(3,2,3)
plot(timelimitc)
title('c) A sparse O(N^2)')

subplot(3,2,4)
plot(timelimitd)
title('d) A full O(N^4)')
hold off

% Darstellung der Saite
[m,n] = size(a);
subplot(3,2,5:6)
for l=1:n;
    plot(a(:,l))
    title('Auslenkung der Saite')
    axis([-inf inf min(min(a)) max(max(a))])
```

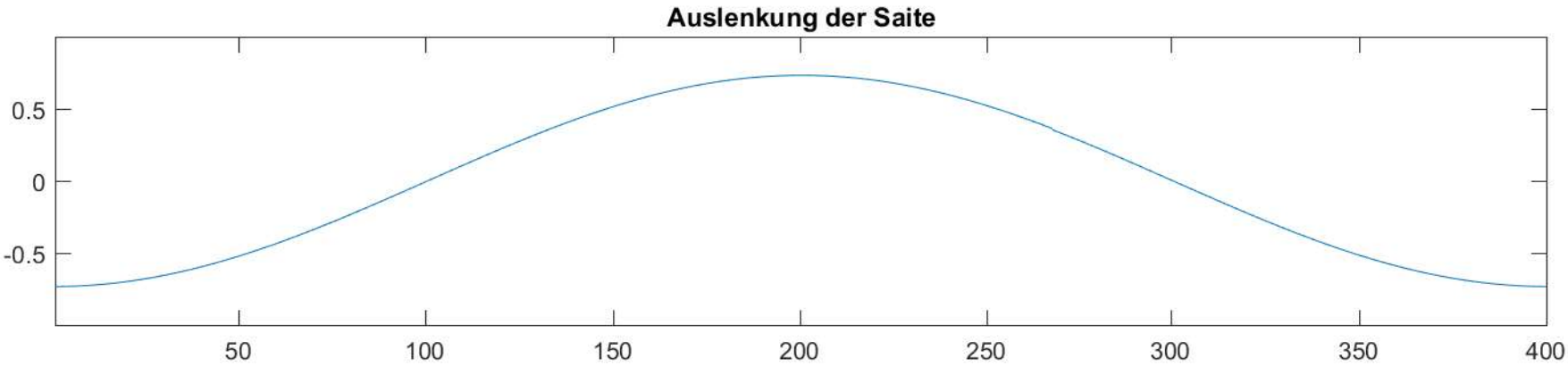
```
    pause(0.1)
end

% anhand der Grafiken kann man schließen, dass die Aussagen ""C"" und ""D""
% richtig sind
% Darstellung der Saite
```


a) A=LU sparse O(N)
×10^-3
2
1.5
1
0.5
0
0
5
10
15


b) A=LU full O(N^2)
×10^-6
12
10
8
6
4
2
0
5
10
15


c) A sparse O(N^2)
×10^-6
4
3.5
3
2.5
2
0
5
10
15


d) A full O(N^4)
×10^-9
2
1.5
1
0.5
0
0
5
10
15


Auslenkung der Saite
0.5
0
-0.5
50
100
150
200
250
300
350
400