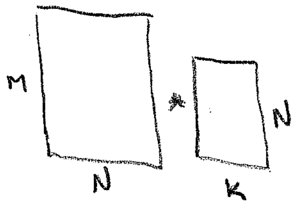
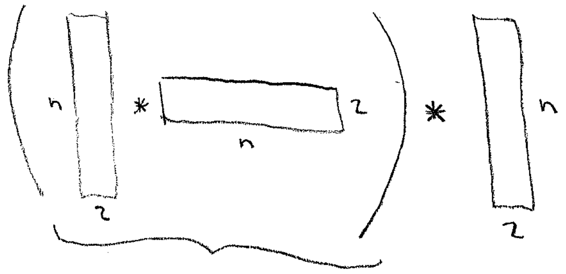
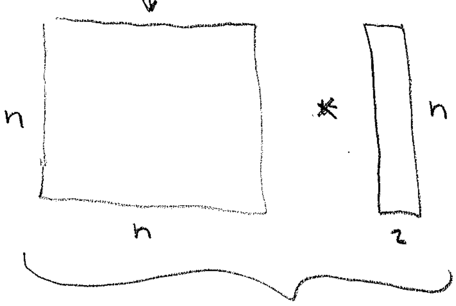
1. Matrix-Matrix Produkt: 

1P

Skript: Asymptotische Komplexität: $\mathcal{O}(MNK)$

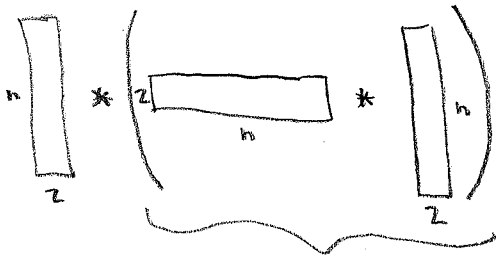
• $(A*B)*C$: 

$\mathcal{O}(2n^2)$ 1P

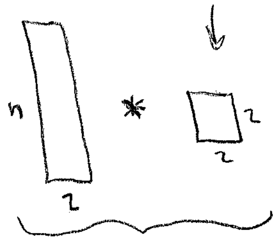


$\mathcal{O}(2n^2)$

$\Rightarrow 2\cdot\mathcal{O}(2n^2) = \mathcal{O}(4n^2) \rightarrow \mathcal{O}(n^2)$ 1P

• $A*(B*C)$: 

$\mathcal{O}(4n)$ 1P



$\mathcal{O}(4n)$

$\Rightarrow 2\cdot\mathcal{O}(4n) = \mathcal{O}(8n) \rightarrow \mathcal{O}(n)$ 1P

Es gilt $(A*B)*C = A*(B*C)$

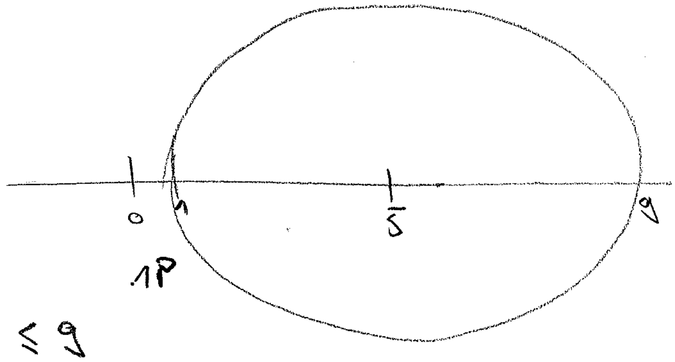
$\Rightarrow$ Es ist viel effizienter $A*(B*C)$ zu berechnen! 1P

6P

2. a) • $\boxed{\text{Lemma } 2.7.4}$ (A strictly diagonal dominant + positive diagonal entries) 1P → 1P

$\Longrightarrow$ A pos. def.

• Gershgorin-circles:



1P

A symm. $\overset{1P}{\Longrightarrow}$ $\lambda_i \in \mathbb{R}$

$\Longrightarrow 1 \leq \lambda_i \leq 9$

$\Longrightarrow$ A pos. def.

b)

```
function Hv = H_mult_v(A,u,v)

    uv = u'*v;
    Hv  = A*v + uv*u;
```

2P

(was, falls andere Implement.?)

If inefficient implementation:
0.5 MATLAB &
2P for correct complexity

$$\left.\begin{array}{l} \cdot\ \vec{u}^T \cdot \vec{v} \longrightarrow \mathcal{O}(n) \\ \cdot\ c\,\vec{u} = uv\,\vec{u} \longrightarrow \mathcal{O}(n) \\ \cdot\ A\vec{v} \longrightarrow \mathcal{O}(n) \quad (n \gg 5) \end{array}\right\} \Longrightarrow \mathcal{O}(n)$$

2P

c)

```
function lmax = dir_pow_meth(A,u,tol)

    % Initialization of variables
    z = u;
    lambda_old = 1e20;
    fertig = 0;

    % Direct power iteration
    while ~fertig
        w = H_mult_v(A,u,z);
        lambda_new = norm(w)/norm(z);
        % alternatively, lambda can be calculated
        % by means of the Rayleigh quotient
        fertig = ( (lambda_old-lambda_new)/lambda_old < tol );
        lambda_old = lambda_new;

        z = 1/norm(w)*w;
    end

    lmax = lambda_new;
```

1P — `w = H_mult_v(A,u,z);`
1P — `lambda_new = norm(w)/norm(z);`
1P — `fertig = ...`
1P — `z = 1/norm(w)*w;`

$z^{(0)} = u$

iterate $\left\{ \begin{array}{l} w := M z^{(k-1)} \\ z^{(k)} := \dfrac{w}{\|w\|} \end{array} \right.$ 1P

$\lambda_n^{(k)} \approx \dfrac{\|M z^{(k)}\|}{\|z^{(k)}\|} = \dfrac{\|w\|}{\|z^{(k)}\|}$ 1P

$\lambda_n = \rho_M(z) = \dfrac{z^T M z}{z^T z}$

3. $$M := \begin{array}{cc} \left(\begin{array}{c|c} A & B^T \\ \hline B & 0 \end{array}\right) & \begin{array}{l} \}\, n \\ \}\, m \end{array} \\ \begin{array}{cc} \underbrace{\qquad\qquad}_{n} & \underbrace{\quad}_{m} \end{array} & \end{array}$$

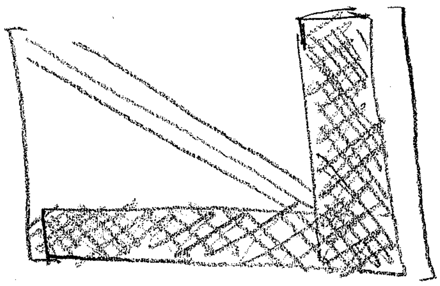
a) M pos. def. $\iff v^T M v > 0 \quad \forall v \in \mathbb{R}^{n+m}$ 1P

Counterexample: $v := \begin{pmatrix} 0 \\ \hline v_m \end{pmatrix}$ 1P $\rightarrow$ Not s.p.d. if zeros on diagonal.

$$\Rightarrow v^T M v = \left(0 \mid v_m^T \right) \left(\begin{array}{c|c} A & B^T \\ \hline B & 0 \end{array}\right) \begin{pmatrix} 0 \\ \hline v_m \end{pmatrix}$$

$$= \left(0 \mid v_m^T \right) \begin{pmatrix} B^T v_m \\ \hline 0 \end{pmatrix} = 0$$

b) Envelope of M:



2P

−1P, if missing

$\#\text{env}(M) \leq m^2 + 2nm + 3n - 2$

1P

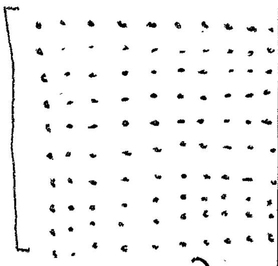
c)

```
% Script to visualize the non-zero pattern of
% the inverse of a tridiagonal matrix A

n = 10;
A = gallery('tridiag',ones(n-1,1),3*ones(n,1),ones(n-1,1));
spy(inv(A));
```

1P 1P

$A^{-1} =$



1P

d) $$A y + B^T z = c \quad (1)$$
$$B y = 0 \quad (2)$$

$$\xRightarrow{A^{-1}\cdot(1)} \quad y + A^{-1} B^T z = A^{-1} c \quad (3) \quad 1P$$

$$\xRightarrow{B\cdot(3)} \quad B y + B A^{-1} B^T z = B A^{-1} c \quad 1P$$

$$\xRightarrow{(2)} \quad \boxed{\underbrace{B A^{-1} B^T}_{=S} z = \underbrace{B A^{-1} c}_{=q}} \quad 1P$$

$$\left.\begin{array}{cc} A & B^T \\ B & 0 \end{array}\right| \begin{array}{c} c \\ 0 \end{array} \Rightarrow \left.\begin{array}{cc} A & B^T \\ 0 & -BA^{-1}B^T \end{array}\right| \begin{array}{c} c \\ -BA^{-1}c \end{array}$$

e)

```
function S = Schurcomplement(d,r,B)

n=length(r);
d_l = [d(1:n-1); 0];
d_u = [0; d(1:n-1)];

A = spdiags([d_l r d_u], -1:1, n, n);

S = B*(A\B')
```

2P (assembly of A)

2P (S = B*(A\B'))

f)

$B' = \underbrace{\square}_{m \times n}$ 1P

$C := A \backslash B' \Rightarrow O(mn)$

↳ dense m×m matrix, because A^{-1} may be dense

$B \cdot C = \Rightarrow O(m^2)$

↳ Cost for single entry $O(1)$

g)

```
function z = solveSchurcomplement(d,r,B,c)

    % assembling A
    n=length(r);
    d_l = [d(1:n-1); 0];
    d_u = [0; d(1:n-1)];

    A = spdiags([d_l r d_u], -1:1, n, n);

    % calculation of q
    q = B*(A\c);

    % An efficient function for multiplication
    Schur_times_vector = @(v)  B*(A\(B'*v));

    % The CG-solver
    z = pcg( @(v) Schur_times_vector(v), q);
```

1P (assembling A)

1P (calculation of q)

2P (efficient multiplication function)

2P (CG-solver)

4.

t_i						
a_i						

$$a(t) = c_1 e^{-\lambda_1 t} + c_2 e^{-\lambda_2 t}$$

a) $\vec{x} = (c_1, c_2, \lambda_1, \lambda_2)$ 1P vector of unknowns

$$F_i(\vec{x}) = a(t_i) - a_i = c_1 e^{-\lambda_1 t_i} + c_2 e^{-\lambda_2 t_i} - a_i \quad 2P$$

["Ansatz-function" evaluated at t_i minus measured values should be as 0 as possible]

b) $\boxed{\vec{x}_{k+1} = \vec{x}_k - \vec{s}, \qquad \vec{s} = \underset{\vec{h} \in \mathbb{R}^n}{\operatorname{argmin}} \| \vec{F}(\vec{x}_k) - J\vec{F}(\vec{x}_k)\vec{h} \|}$ 2P

linear least squares problem in every step

$$JF(\vec{x}_k) = \begin{pmatrix} \vdots \\ \text{grad } F_i(\vec{x}_k)^T \\ \vdots \end{pmatrix} \quad (\text{Jacobian})$$

$$\text{grad } F_i(\vec{x}_k) = \begin{pmatrix} \partial c_1^k F_i(\vec{x}_k) \\ \partial c_2^k F_i(\vec{x}_k) \\ \partial \lambda_1^k F_i(\vec{x}_k) \\ \partial \lambda_2^k F_i(\vec{x}_k) \end{pmatrix} = \begin{pmatrix} e^{-\lambda_1^k t_i} \\ e^{-\lambda_2^k t_i} \\ -t_i c_1^k e^{-\lambda_1^k t_i} \\ -t_i c_2^k e^{-\lambda_2^k t_i} \end{pmatrix}$$

2P

Note: 4b has to be graded w.r.t. 4a!

c)

```
% Name:
% Problem 4, run-script

format long;

% Gauss-Newton driver
clear all;
close all;

% load the variables ti and ai
load activities.mat;

% determine the parameters c and lambda
[c,lambda] = lsq_gauss_newton(ti,ai)

at = c(1).*exp(-lambda(1).*ti) + c(2).*exp(-lambda(2).*ti);

% plot measured data and least squares optimized curve
figure(2), clf
plot(ti,ai,'.r')
hold
plot(ti,at);
xlabel('t')
ylabel('a(t)')
```

$t_1 = 0 \Rightarrow a_1 = c_1 + c_2 \Rightarrow c_1 = a_1 - 2$

$t_2: \quad a_2 = c_1 e^{-\lambda_1 t_1} + c_2 \quad \Rightarrow \lambda_1 = - \frac{\ln\left(\frac{a_2 - c_2}{c_1}\right)}{t_2}$

```
% Name    :
% Problem 4, 'lsq_gauss_newton'

function [c,lambda] = lsq_gauss_newton(t,a)

    % estimate c1, lambda1
    c1      = a(1) - 2
    lambda1 =  -log((a(2)-2)/c1)/t(2)

    x = [ c1; 2; lambda1; 0 ];

    err = [];
    not_converged = true;

    % Gauss-Newton iteration
    while( not_converged )

        F  = x(1)*exp(-x(3).*t) + x(2)*exp(-x(4).*t) - a;

        JF = [ exp(-x(3).*t), ...
               exp(-x(4).*t), ...
               -t.*x(1).*exp(-x(3).*t), ...
               -t.*x(2).*exp(-x(4).*t) ];

        s = JF\F;
        x = x - s

        err = [err; norm(s,2)];
        not_converged = ( err(end) > 1e-10 );
    end


    %  plot error
    semilogy(1:length(err), err, 'b')
    xlabel('iteration')
    ylabel('error (2-norm)')

    c      = [x(1); x(2)];
    lambda = [x(3); x(4)];
```

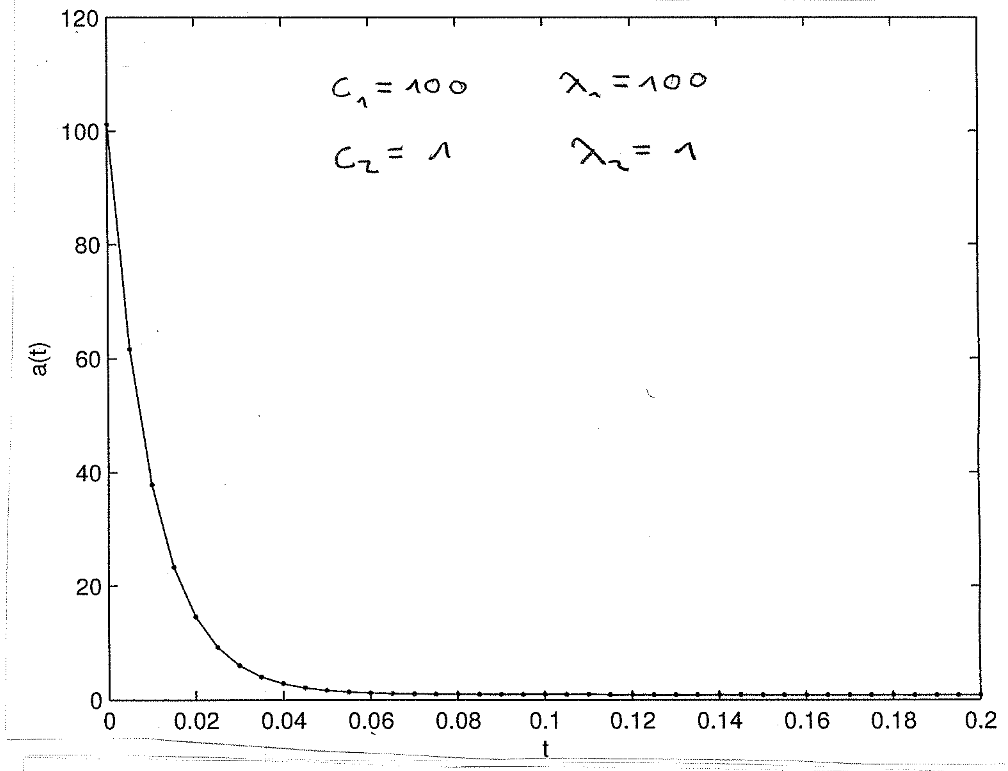
1P (c1), 1P (lambda1), 1P (F), 1P (JF), 2P (s, x), 1Pd) 1P (err, not_converged), 1Pd) (plot error), 1P (c, lambda)

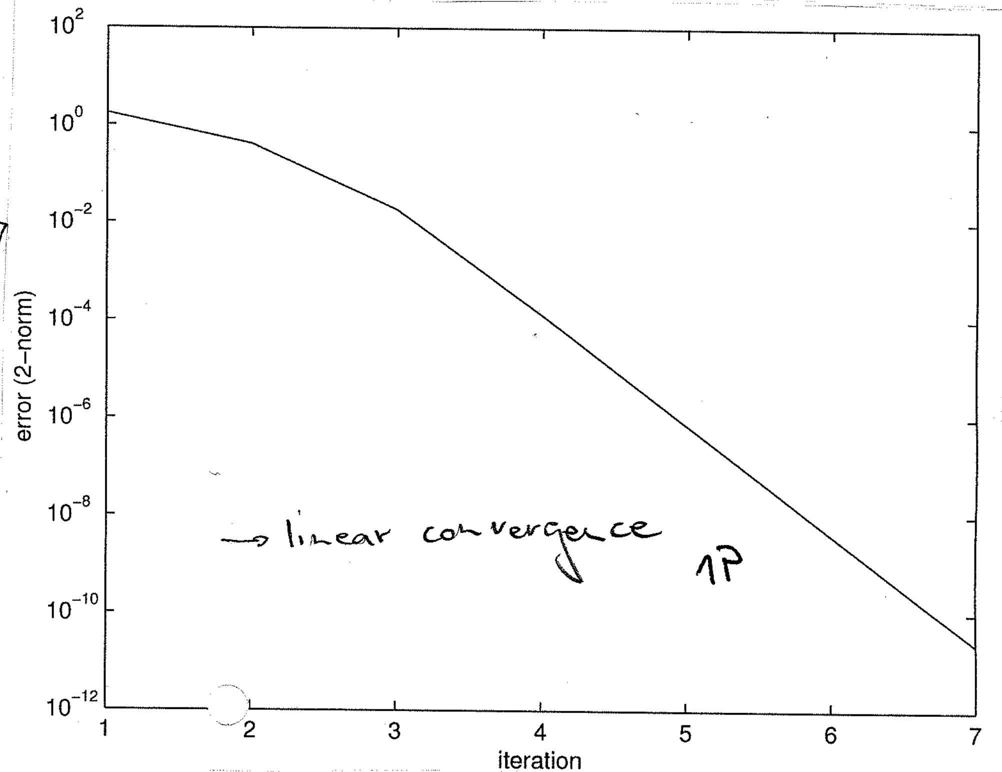
? Not the error

err = [err; norm(x - x exact)]

↑ can be computed by running the code.

[a_i = a(t).* exp(0.0001 * randn(51,1))]





5. a) $\dddot{u} = -\sin(\dot{u}) + g(y)$ $(*1)$

$\vec{y} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} := \begin{pmatrix} u \\ \dot{u} \\ \ddot{u} \end{pmatrix}$ 1P $\Rightarrow$ $\boxed{(*1) \iff \begin{aligned} \dot{y}_1 &= y_2 \\ \dot{y}_2 &= y_3 \\ \dot{y}_3 &= -\sin(y_2) + g(y_1) \end{aligned}}$ 2P

$\dot{\vec{y}} = \vec{f}(\vec{y})$

b)

$$\begin{array}{c|ccc} c_1 & & & \\ \vdots & a_{2,1} & \ddots & \\ \vdots & \vdots & & \\ c_s & a_{s,1} & \cdots & a_{s,s} \\ \hline & b_1 & \cdots & b_s \end{array}$$

$\rightarrow$ Explicit s-stage Runge-Kutta single step meth. 1P

$c_i = \sum_{j=1}^{i-1} a_{ij}$, $i,j = 1,\dots,s$

$t_{j+1} = t_j + h$

$\vec{k}_i := f(\vec{y}_0 + h \cdot \sum_{j=1}^{i-1} a_{ij} \cdot \vec{k}_j)$ $i = 1,\dots,s$ 1P

$y_1 := y_0 + h \sum_{i=1}^{s} b_i \vec{k}_i$

here: $\vec{k}_1 = \vec{f}(\vec{y}_0)$

$\vec{k}_2 = \vec{f}(\vec{y}_0 + \frac{1}{3} h \vec{k}_1)$ 1P

$\vec{k}_3 = \vec{f}(\vec{y}_0 + \frac{2}{3} h \vec{k}_2)$

$\vec{y}_1 = \vec{y}_0 + \frac{1}{4} h \vec{k}_1 + \frac{3}{4} h \vec{k}_3$

```
% Name :
% Problem 5, run script

% function handle for g(u)
g = @(u) -u;


% Initial condition etc.
y0 = [1; 0; 0];
T  = 10;
N  = 100;

% call the heun_integrator_driver
[t, y] = heun_driver(g, y0, T, N);

% plotting the solution
plot(t,y)
```

d)

```
% Name :
% Problem 5, 'heun_driver'


function [t, y] = heun_driver(g, y0, T, N)
                                                              2P
    rhs = @(y) [ y(2); y(3); -sin(y(2))+g(y(1))];

    [t, y] = heun_integrator(rhs, y0, T, N);  1P
    %[t, y] = ode45(rhs,[0 T], y_init);
end
```

c)

```
% Name :
% Problem 5, 'heun_integrator'

function [t, y] = heun_integrator(odefun, y0, T, N)

    t = 0;
    y = y0';
                               1P
    h = T/N;

    y_old = y0;

    % main integration loop
    for l = 1:N

        % Butcher-tableau
        k1 = odefun(y_old          );
        k2 = odefun(y_old + 1/3*h*k1);
        k3 = odefun(y_old + 2/3*h*k2);           1P

        y_new = y_old + 1/4*h*k1 + 3/4*h*k3;

        t = [t; t(end)+h];
        y = [y; y_new'  ];                  1P

        y_old = y_new;

    end
end
```