Wiederholung Angewandte Informatik

ZENTRALE THEMEN FÜR DIE KLAUSUR

* Funktionen schreiben, Parameterübergabe, ...
* Vergleiche (if else), Bedingungen, logische Verknüpfungen (&&, ||, ~)
* Schleifen, auch Doppelschleifen (for, while), Summe, Produkt
* Arrays durchsuchen bzw. erzeugen: :-Operator, size, length, min, max, x.^2, A(m,n) = ...
* plot, subplot, hold on, ...
* OOP: Klassen erstellen, Eigenschaften, Konstruktor, Methoden
* Text-Dateien: lesen und schreiben, Text in Zahlen wandeln, ASCII-Werte
* Fourier-Synthese, Sample-Rate, FFT, audioread, …
* Bilder einlesen, RGB-Daten analysieren, ...
* Polynom-Fit, Analyse der Parameter p(n) , roots, polyder, ...
* DGL lösen zu AW, Parameter-Übergabe an DglFnc, DGL 2. Ordnung in System 1. Ordnung, DGL-Ergebnis interpretieren: x = y(:,1), v = y(:,2), min, max, mean, std, …

Wichtige Befehle und Operatoren

KAPITEL 2:

function, ==, <, >, &&, ||, ~, sqrt, sin, cos, abs, exp, if, else, switch, case,

for, while, fprintf, input, length, size, min, max, :-Operator, linspace, rand, randn

plot, surf, subplot, hold, struct, .-Operator, return, break

fopen, fclose, fgetl, fprintf, sprintf, strcmp, str2double, num2str, ischar, char, ASCII-Wert

xlsread, xlswrite (Excel), save, load (MAT-Files)

classdef, properties, methods, Rückgabe des Objekts oder von anderen Daten

KAPITEL 3:

uiputfile, uigetfile, uisetcolor

KAPITEL 4:

fft, audioread, imread, imwrite, image

polyfit, polyval, mean, std, max, min, sort, roots, fzero, integral

ode45, \-Division

FÜR KLAUSUR NICHT RELEVANT:

- cell, cell2mat
- Array-Vorbelegung: zeros, ...
- meshgrid
- sscanf
- varargs
- try ... catch
- Graphic Handles
- GUI, bis auf Std-Dialoge
- MEX-C, Simulink, …
- COM/OLE
- Internet, Java, JVM, HTML, JavaScript
- USB, Arduino

Wichtige Verfahren (1)

1. Analyse von Feldern

1-dim. Array *v*:

```
anz = length( v );
```

mit Einfachschleife durch die Elemente:

```
for( n = 1:anz )
  wert = v(n) ;      % Analyse – Auslesen der v-Daten
  v(n) = wert;       % Array-Elemente erzeugen
```

2- oder 3-dim.Array *A*:

```
sz = size(A); anzZ = sz(1); anzS = sz(2);
```

Doppelschleife:

```
for( z = 1:anzZ )
  for( s = 1:anzS )
    wert = A(z,s);       % Analyse – Auslesen der A-Daten
    A(z,s) = wert;       % Array-Elemente erzeugen
```

Wichtige Verfahren (2)

2. Summenbildung

Aufsummieren von Termen *a*(*n*):

Summe = Σ *a*(*n*)

mit Schleife die Elemente aufsammeln:

```
Summe = 0;          % Topf für Summe, am Anfang leer
for( n = 1:anz )
  term = a(n);              % nächsten Term bestimmen
  Summe = Summe + term;     % Terme in Summen-Topf
```

Summe von Funktionen (Fourier-Reihe oder Taylor-Reihe, vgl. Labor 12):

```
y = 0;          % Topf für Summe, am Anfang leer
for( n = 1:anz )
  term = A(n)*sin( 2*pi*n*t );  % nächsten Term bestimmen
  y = y + term;                 % in Summen-Topf
```

Anwendung: Taylorreihe (Klausur WS 16/17), siehe auch e-Funktion

Funktion `y = taylor( x, nMax )`

Aufsummieren von Termen *a*(*n*): $y = \Sigma\, a(n)$

$$y = \sum_{n=0}^{nMax} (-1)^n \, (n+1) \, x^n$$

Verfahren:

```
Summe = 0;          % Summe, am Anfang leer
for( n = 1: nMax )
  term = a(n);              % nächster Term
  Summe = Summe + term;     % Terme zu Summe
```

Lösung:

```
function y = taylor( x, nMax )
  y = 0
  for( n = 0:nMax )
     y = y + (-1)^n * (n+1) * x.^n;
  end
```

Anwendung: Fourierreihe (Klausur SoSe 16)

Funktion `fourier( f, tm, SR, d )`

Aufsummieren von Termen *a*(*n*): $y = \Sigma\, a(n)$ $\qquad y(t) = \sum_{n=1}^{?} A(n)\,\sin(2\,\pi\,n\,f\,t)$

Die Amplitude *A*(*n*) des n-ten Tons ist 1/n, für *n*=1 bis zu dem n-Wert, für den der Betrag von *A*(*n*) kleiner wird als die übergebene Schranke *d*. Zeichnen Sie den zeitlichen Verlauf des Samples.

Verfahren für unbekannte Zahl von Durchläufen:

```
weiter = 1;
n = 0;
while( weiter )
  n = n + 1;
  term = a(n);                % nächster Term
  if( abs(term) < Schranke )
    weiter = 0;  . . .
```

Lösung: Fourierreihe (Klausur SoSe 16)

```
function fourier( f, tm, SR, d )
   t = 0:1/SR:tm;
   weiter = 1;
   n = 1;  y = 0;
   while( weiter )
      An = 1/n;
      if( abs( An ) < d )
        weiter = 0;
      else
        y = y + An * sin( 2*pi*n*f*t );
        n = n + 1;
      end
   end
   plot( t,y )
end
```

Wichtige Verfahren (3)

3. Iteration

Sukzessive Berechnung von Termen $x(n+1)$ aus vorherigen Termen

$x(n+1)$ = Funktion($x(n)$);

mit Schleife die Terme iterativ berechnen:

```
xn = x0;          % Term am Anfang auf Startwert
for( n = 1:anz )      % oder auch while-Schleife für Abbruch
  diff = . . . ;      % Differenz zum nächsten Term bestimmen
  xn = xn + diff;     % xn um diff weiterführen
```

Beispiele:

Newton-Verfahren:

```
diff = - f0( xn ) / f1( xn );       % Differenz = Funktion / Ableitung
```

Eulersches Polygonzug-Verfahren:

```
diff = h * DglFun( tn, xn, par );  % Differenz = h * Ableitung
```

Wichtige Verfahren (4)

4. Schleifenabbruch

Bedingung im Kopf einer while-Schleifen: `while( bedingung )`

Die Schleife wird so lange durchlaufen, wie die *bedingung* erfüllt ist.

Beispiele:

Vergleich mit berechnetem Wert (muss zu Beginn vorbelegt sein):

```
wert = -1;
while( wert < 0 )
  wert = input( 'Zahl eingeben: ' );
```

Explizites „Flag“ (z.B. Variable *weiter*) zum Steuern der Durchläufe:

```
weiter = 1;
while( weiter )      % Werte ungleich 0 sind immer wahr
   t = fgetl( fid );
   if( t == -1 )
     weiter = 0
```

Wichtige Verfahren (5)

5. Array-Daten übertragen

Aufgabe: Daten aus einem Array *A* (vollständig oder teilweise) in Array *B* übertragen.

```
for( n = n1:n2 )       % Elemente von Startindex n1 bis Endindex n2
   B(n) = A( f(n) );   % zugeordnetes Element f(n) verwenden
```

Beispiele:

Bilddaten aus *A* an vertikaler Achse spiegeln:

```
x = anz:-1:1;       % z.B.: anz = 5;  =>  x = [5,4,3,2,1];
for( s = 1:anz )    % durch alle Spalten s
  B(:,s,:) = A(:,x(s),:);   % Zeilen und Farben unverändert
```

oder

```
B = A(:,anz:-1:1,:);   % Zeilen und Farben unverändert
```

Detail aus Bild *A*, nur von Zeile *z1* bis Zeile *z2*:

```
B = A(z1:z2,:,:);           % Spalten und Farben unverändert
```

Wichtige Verfahren (6)

6. Array durchsuchen

Aufgabe: Maximalen Wert in Feld y suchen (analog Funktion *dataPlot*)

```
for (n = 1:num )
  y(n) = input( 'Zahl: ' ); % y-Vektor: eingelesene Werte
end
```

Mehrere Methoden für Maximum:

Einfachste Variante: sich am Ende sich das Maximum geben lassen

```
mx = max( y );
```

Andere Variante: in der Schleife abfragen, ob Wert größer als bisheriger Wert

```
for (n = 1:num ) ... if( y(n) > mx )  mx = y(n);  end
```

Problem: Wie behandelt man den 1. Wert?

```
mx = -Inf;  % am Anfang mx mit -Unendlich vorbelegen
```

Oder in der Schleife 1. Fall separat behandeln:

```
if( n == 1 ) mx = y(1);
else if(y(n) > mx )  mx = y(n);  end
end
```

Wichtige Verfahren (7)

7. Klassen *(Beispiel aus der Klausur WS 2015/16)*

Entwerfen Sie die Klasse *FitData*, mit den Eigenschaften *x*, *y* und *yFit*.

Der Konstruktor hat als Eingabeparameter eine Anzahl von Messpunkten als x- und y-Vektoren und speichert die Daten in den Eigenschaften *x* und *y*. Zusätzlich besitzt die Klasse drei Methoden:

Die Methode *polyfit*(…,fitGrad), mit Übergabe des Fit-Grades, bestimmt einen Polynomfit der x-y-Daten und berechnet danach zu den x-Werten die Funktionswerte des Polynoms, die unter *yFit* abgespeichert werden.

Die Methode *plotAll* plottet sowohl die Messpunkte als auch die Ausgleichskurve in einer Zeichnung.

Die Methode *statistics* berechnet den Mittelwert und die Standardabweichung der y-Werte und liefert die beiden Zahlen als Rückgabewerte der Methode.

Wie lauten die folgenden Aufrufe im Command-Window:

Konstruktor-Aufruf, Aufruf der drei Methoden?

Beispiel Klasse fitData

***Klasse mit zwei* Bereichen:**
Eigenschaften und Methoden

```
classdef fitData
  properties
    x = [];
    y = [];
    yFit = [];
  end
```

Aufrufe im Command Window:

```
>> o = fitData( [1 2 3], . . .
                [2 1 4] );
>> o = o.polyfit( 1 );
>> [m,s] = o.statistics;
```

```
methods
    function obj = fitData( x, y )
      obj.x = x;
      obj.y = y;
    end

    function obj = polyfit( obj, grad )
      pol = polyfit( obj.x, obj.y, grad);
      obj.yFit = polyval( pol, obj.x );
    end

    function [mw,sa] = statistics( obj )
      mw = mean( obj.y );
      sa = std( obj.y );
    end
  end
end
```

Rückgabe der Methoden:

- Geändertes Objekt *obj*
- Andere Werte, z.B. *mw*

Probeklausur 2015

Aufgabe 1: Klasse Polynom

OOP: Klassendefinition, Eigenschaften, Methoden, Polynome: roots, polyval

Aufgabe 2: Funktion spiegelBild

Bilder, 2-dim. Array, size, Doppelschleife

Aufgabe 3: Funktion lnReihe,

Summe, Funktion aufbauen, while-Schleife, abs, Hochzählen

Aufgabe 4: Funktion mechode

DGL 2. Ordnung, x = y(1), v = y(2)

Aufgabe 5: Funktion intervall

Feld durchsuchen, if, &&-Verknüpfung

Probeklausur

Aufgabe 1: Klasse Artikel

OOP: Klassendefinition, Eigenschaften, Methoden

Aufgabe 2: Funktion Planet(m, L, k, x0, v0, tm)

DGL 2. Ordnung, Übergabe von Parametern an DGL-Funktion, min, max, plotten

Aufgabe 3: Funktion num = pos2D(A),

2-dim. Array durchsuchen, size, Doppelschleife, A(z,s), zählen (Summe)

Aufgabe 4: Funktion FourierSynth(n_max)

Fouriersynthese, Sample-Rate, :-Operator, FFT, plotten, subplot

Aufgabe 5: Funktion k = Federkonst (m, x)

Polynom-Fit, Interpretation der Koeffizienten polyval

Wichtige Aufgaben aus dem Labor / aus der Vorlesung

* Alle Aufgaben aus Labor 11
* Labor 10: LoeseDgl1, LoeseDgl2 / DglSolver, DglFun, Solve_Feder_<n>, Fun_Feder_<n>
• Labor 9: checkParabel, temperaturen, interpolExcel / kurvendisk, ausgleichsgerade, FallParameter, datenauswertung
* Labor 8: calcArea, createImage / farben, grauFilter, rectImg, detailBild, Schwerpunkt
* Labor 7: signal5, xlsAmplitudes, xlsFFT / rechteck, SignalArray, spektrum
* Labor 5: save_person, leseExcel / fKubik, write_file, read_file
* Labor 4: class Auto, class geom, class rect
* Labor 3: fncPlot, dataPlot, Biegelinie2, DynGrundgesetz / einlesen, countNeg, statistik, sinPlot, sinPlot2, bmi, polyPlot, view3D
• Labor 2: kubik, Mittelwert, Gr_1x1, Biegelinie, intervall / quadrat, summe, produkt, ein_mal_eins, ListeFTM, posnum, quadrat30
• Labor 1: calcJ, warnResonance, checkEmission / Mult, wahl, GedSchwingung, checkInput

Labor 10 - DGL

Aufgabe LoeseDgl1

```
function [mw,sa] = LoeseDgl1( x0, tm, c )
  [t,x] = ode45( @dglFcn1, [0,tm], x0, [], c );
  mw = mean( x );
  sa = std( x );
  plot( t,x )

function dx_dt = dglFcn1( t,x, c )
  dx_dt = - 2 * c * x.^3 + sin( pi*t );
```

Aufgabe LoeseDgl2

```
function LoeseDgl2( x0, v0, tm )
   [t,w] = ode45( @DglFnc2, [0,tm], [x0,v0] );
   x = w( :, 1 );
   plot( t, x );

function dy_dt = DglFnc2( t, y )
   dy_dt(1,1) = y(2);
   dy_dt(2,1) = 3 / y(1)^3 - 5 * y(2);
```

Variablennamen

meistens: *x*, *y*, *z*
Erdanziehung: *r* (Radius)
Zerfall: *n* (Teilchenzahl)
Nav.-St.: *v*, *u* (Geschwindigkeit)
El.-Magn.: *E*, *B* (el./magn. Feld)

Labor 9 - Mathematik, Polynome, Polynom-Fit (1)

Wichtige Aufgaben: checkParabel, temperaturen, interpolExcel

Aufgabe checkParabel

```
function checkParabel( a, b, c, x1, x2 )
  pol = [ a, b, c ];          % Koeffizienten des Polynoms
  nullst = roots( pol );      % Nullstellen
  d1 = polyder( pol );        % Ableitung und Extremum
  extr = roots( d1 );
  x = linspace( x1, x2, 100 ); % x-y-Werte
  y = polyval( pol, x );
  hold on                     % Funktion zeichnen
  plot( x, y, 'b-' );
  if( isreal( nullst(1) ) )     % reelle Nullstelle ?
    plot( nullst(1), 0, 'gs' );  % Einzeichnen
  end
  ...
```

Labor 9 - Mathematik, Polynome, Polynom-Fit (2)

Aufgabe temperaturen

```
function  [mw,sf] = temperaturen()
   werte = [16.2, 15.8, 17.6, 19.1, 19.2, 21.0, 20.9, 22.5, 23.1, 23.7];
   t = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

   p = polyfit( t, werte, 1 );   % Ausgleichgerade bestimmen

   w = polyval( p, t );          % Werte auf der Ausgleichgeraden
   hold on
   plot(t,w);                  % Ausgleichgerade zeichnen
   plot(t,werte,'s');          % Messpunkte zeichnen
   hold off

   % Mittelwert und Standardfehler
   mw = mean( werte );
   sa = std( werte );
   sf = sa / sqrt( length(werte) );
```

Labor 8 - Bilder, Arrays (1)

Wichtige Aufgaben: calcArea, createImage

Aufgabe calcArea

```
function p = calcArea()
   f = uigetfile( '*.bmp', 'Bild wählen' );
   if( f == 0 )  return;  end
   A = imread( f ); % Datei laden
   sz = size( A ); % Größe bestimmen
   nZ = sz(1);  nS = sz(2);
   nBody = 0;          % Zahl der Punkte im Körper mit 0 vorbelegen
   for( z = 1:nZ )     % alle Pixel untersuchen (Zeilen und Spalten)
     for( s = 1:nS)
        col = sum( int32( A(z,s,:) ) ); % Check Farbe nahezu schwarz
        if( col < 100 )
           nBody = nBody + 1;          % hochzählen
        end
     end
   end
   f_bild = nZ*nS;   p = nBody / f_bild * 100; % prozentuale Fläche
   image( A ); axis image  % Bild zeichnen
```

Labor 8 - Bilder, Arrays (2)

Aufgabe createImage

```
function createImage()
   for( z=1:255 )       % alle Zeilen und Spalten definieren
      for( s=1:255 )
         A(z,s,1) = z;
         A(z,s,2) = s;
         A(z,s,3) = 100;
      end
   end
   A = uint8(A);        % Daten in uint8 wandeln, zur Sicherheit

   % Bild anzeigen
   image( A );        axis image;

   % Dateinamen zum Speichern holen
   f = uiputfile( 'createImage.bmp' );
   if( f == 0 ) return;  end

   imwrite( A, f )    % Daten abspeichern
```

Labor 7 - Musik, Fourier-Reihen, FFT (1)

Wichtige Aufgaben: signal5, xlsAmplitudes, xlsFFT

Aufgabe signal5

```
function signal5( f0, tm, SR )
    t = 0 : 1/SR : tm;
    % Array y mit Funktionswerten an den t-Stützpunkten
    y = sin(2*pi*f0*t) + 1/5*sin(2*pi*2*f0*t) +  1/4*sin(2*pi*3*f0*t) - ...
        1/4*sin(2*pi*4*f0*t) + 1/3*sin(2*pi*5*f0*t);
    % Ausgabe der Funktion als 2D-Plot
    plot( t, y );
    % fester Ausgabebereich
    tp = 10 / f0;
    yp = 1.1 * max( y );
    axis( [0 tp -yp yp] )
```

Labor 7 - Musik, Fourier-Reihen, FFT (2)

Aufgabe xlsAmplitudes

```
function [t,y] = xlsAmplitudes( filename, f0, tm, SR )
  % Excel-Datei einlesen
  A = xlsread( filename );
  anz = length( A );     % Anzahl der Amplituden
  t = 0:1/SR:tm;         % Zeit-Intervall
  % Beiträge aufsummieren
  y = 0;
  for( n=1:anz )
    y = y + A(n)*sin(2*pi*n*f0*t);
  end
```

Funktionsaufruf

```
>> [t,y] = xlsAmplitudes( 'dat.xlsx', 440, 2, 40000 )
```

Labor 7 - Musik, Fourier-Reihen, FFT (3)

Aufgabe xlsFFT

```
function xlsFFT( filename )
  data = xlsread( filename );      % Excel-Daten einlesen
  t = data( :, 1 );   y = data( :, 2 );
  len  = length( t );              % Samplerate berechnen
  tmax = t(len);
  SR = len/tmax;
  subplot(2,1,1);            % Ausgabe der Funktion als 2D-Plot
  plot( t, y );
  maxy = 1.1 * max( y );
  axis( [0 tmax/20 -maxy maxy] )   % axis-Beschränkung
  c = fft( y );                    % Fourier-Transformation
  A = abs( c );                    % Betrag der Frequenzamplituden
  ...
  subplot(2,1,2)                   % 2D-Plot der Frequenzamplituden
  plot( A )
```

Labor 5 - Dateien, struct (1)

Wichtige Aufgaben: save_person, leseExcel

Aufgabe save_person

```
function save_person( num_pers, filename )
    for n = 1:num_pers
        person(n).name  = input( 'Name: ', 's' );
        ...
    end
    fid = fopen( filename, 'w' );   % Datei zum Schreiben öffnen
    % Check, ob fid gültig ist
    if( fid < 0 ) ... end
    for( n = 1:num_pers )
       fprintf( fid, 'Name: %s, Tel.Nr.: %g, ', ...
                person(n).name, person(n).telnr  );
       ...
    end
    fclose( fid );
```

Labor 3 - Grafik, Arrays (2)

Aufgabe dataPlot

```
function  dataPlot( num )
  fprintf( 'Bitte nacheinander %d Zahlen eingeben: \n', num )
  % Matrix x und y berechnen,
  for (n = 1:num )
    % y-Vektor: eingelesene Werte
    y(n) = input( 'Zahl: ' );
  end
  % Daten plotten
  plot( y, 'k:s' );
  ...
```

Array y erweitern:

len = length(*y*);

n = *len* + 1;

y(*n*) = *wert*;

In Array y Maximum suchen:

mx = max(y);

Labor 3 - Grafik, Arrays (4)

Aufgabe DynGrundgesetz

```
function [mw,sf] = DynGrundgesetz( t, tv )
   n  = length( t );
   nv = length( tv );
   if(n ~= nv ) ... end     % Check der Längen
   L = 0.15; % m

   v = L ./ tv;
   a = v ./ t;

   plot( t, v, '-b', t, a, '-r' )
   legend( 'v = v(t)', 'a = a(t)' );
   xlabel( 't/s' );
   ylabel( 'v / m/s | a / m/s^2' )

   mw = mean( a );
   sa = std( a );
   sf = sa / sqrt(n);
```

Labor 2 - Schleifen, Doppelschleifen, Bedingungen (1)

Wichtige Aufgaben: kubik, Mittelwert, Gr_1x1, Biegelinie, intervall

Aufgabe kubik

```
function  kubik(m,k)
   ...
   for( n=m:k )
      q = n * n * n;
      fprintf( '  %3d  |  %3d   \n', n, q );
   end
```

Aufgabe Mittelwert

```
function mw = Mittelwert( n )
   s = 0;   % Summe mit 0 vorbelegen
   for( k=1:n )
      x = input( 'Nächste Zahl: ' );
      s = s + x;
   end
   mw = s/n;
```

Vorbelegung Rückgabewert

hier z.B.: `mw = 0;`

andere Möglickeiten:

`mw = NaN;`

Vergleich: `if( isnan(mw) ) ...`

`mw = -Inf;`

Vergleich: `if( x > mw ) ...`

Labor 2 - Schleifen, Doppelschleifen, Bedingungen (2)

Aufgabe Gr_1x1

```
function  Gr_1x1()
   for( m = 11:20 )
      for( n = 1:10 )
         erg = m*n;
         fprintf( ' %3.0f ', erg );
      end
      fprintf( '\n');
   end
```

Aufgabe Biegelinie

```
function Biegelinie()
   ...
   for( F = 0 : 5 : 20 )
      for( x = 0.00 : 0.15 : 0.45 )
         w = 1e3 * F / ( 48*E*I ) * x * ( 3*L^2 - 4*x^2 );  % in mm
         fprintf( '%5.1f | ', w );
      end
```

Labor 2 - Schleifen, Doppelschleifen, Bedingungen (3)

Aufgabe intervall

```
function z = intervall( m, n )
    z = m-1;    % Vorbelegung mit Start-Zahl
    anz = 0;    % Vorbelegung der Summe
    while ( z < m || z > n )
        if( anz >= 5 )
            z = NaN;
            fprintf( 'Zu viele fehlerhafte Eingaben. \n' );
            return;
        end
        fprintf( 'Zahl zwischen %g und %g: ', m, n );
        z = input( '' );
        anz = anz + 1;
    end
    fprintf( 'Die korrekte Zahl ist %g. \n', z );
```

Labor 1 - Funktionen, Verzweigungen, Bedingungen (1)

Wichtige Aufgaben: calcJ, warnResonance, checkEmission

Aufgabe calcJ

```
function J = calcJ( m, ra, ri )
  J = m/2* ( ra^2 + ri^2 );
```

Aufgabe warnResonance

```
function warnResonance( fext, fres )
  d = fext - fres;
  dr = abs( d ) / fres;
  if( dr < 0.1 )
     fprintf( 'fext = %g Hz nahe an Resonanz %g Hz \n', fext, fres );
  else
     fprintf( 'Kein Problem mit Resonanz \n' );
  end
```

Labor 1 - Funktionen, Verzweigungen, Bedingungen (2)

Aufgabe checkEmission

```
function checkEmission( lambda )
   % Cd (520 - 530) nm
   if( lambda >= 520 && lambda <= 530 )
      fprintf( 'Wellenlänge %g nm passt zu Cd \n', lambda );
      return;
   end
   % Hg (545 - 580) nm
   if( lambda >= 545 && lambda <= 580 )
      fprintf( 'Wellenlänge %g nm passt zu Hg \n', lambda );
      return;
   end
   ...
   fprintf( 'Wellenlänge %g nm passt zu keinem Element \n', lambda );
```

Labor 12 - Wiederholung

Aufgabe 1: Funktion [t,y] = fourierReihe(A, nf, f, SR, tm)

Fouriersynthese, Sample-Rate, :-Operator, Arrays

Aufgabe 2: Funktion num = bild100(fn_in, fn_out)

Bilder, RGB, 2-dim. Array durchsuchen, size, Doppelschleife, A(z,s) , logische Verknüpfungen

Aufgabe 3: Funktion num = calcRoots(pol)

Polynome, length, roots, isreal

Aufgabe 4: Funktion [a,w0,phi0] = fit2(t, phi)

Polynom-Fit, Interpretation der Koeffizienten polyval

Aufgabe 5: Funktion x_max = dgl(k, tm, x0, v0)

DGL 2. Ordnung, Übergabe von Parametern an DGL-Funktion, min, max, plotten

Aufgabe 6: Funktion [mw,sf] = stdFehler(xData)

Dateien lesen, Text in Zahlen wandeln, Vektoren, .-Operator, sum

Labor 12 – Wiederholung (2)

Aufgabe 7: Klasse Wochenstunden

classdef, Entwurf von Klassen, Eigenschaften, Methoden, Aufruf (Konstruktor, Methode)

Aufgabe 8: Funktion w = invertiereWort

Analyse eines Feldes, Aufbau eines Feldes, Belegen der Komponenten

Aufgabe 9: Funktion [y,N] = sinhReihe(x, delta)

Aufbau einer Taylor-Reihe, analog wie Fourier-Reihe, Summenbildung