0.1 Problemstellung ODE

$y^n(x) = f(x, y(x), y'(x), ..., y^{(n-1)}(x))$

gesucht: $y = y(x) \implies$ Lösungen im Intervall [a,b]

Bemerkungen:

1. Neben den ODE gibt es noch die partiellen
2. Es gibt Systeme von Diff.Gleichungen, die u.Umständen gekoppelt sind, z.B gekoppelte Schwingungen
3. Allgemeine Lösung: DGL n-ter Ordnung enthält noch n unabhängige Parameter (Integrationskonstante unbestimmter Integrale)
4. Unterscheidung Anfangs- oder Randwertproblem, bei Anwendung numerischer Verfahren

Definition Anfangswertproblem (AWP)

1.Ordnung:

Gesucht: Spezifische Lösungskurve $y = y(x)$, durch vorgegebenen $P = (x_0, y(x_0))$

Gegeben: $y'(x), = f(x, y(x))$ und Anfangswert $y(x_0)$

2.Ordnung:

Gesucht: Spezifische Lösungskurve $y = y(x)$, durch vorgegebenen $P = (x_0, y(x_0))$ und im Punkt x_0 vorgegebene Steigung $y'(x_0) = m$

Gegeben: $y''(x), = f(x, y(x), y'(x))$ und Anfangswerte $y(x_0), y'(x_0)$

0.2 Richtungsfelder DGL 1.Ordnung

Linienelement: Pfeil.

Tangente $y'(x) = f(x, y(x))$ durch Pfeil graphisch anzeigen, diese geben in jedem Punkt, die Richtung der Lösungskurve an.

Idee: Lösung AWP dem Richtungsfeld möglichst genau zu folgen $\implies$ Diskretisierung benötigt:

Intervall [a,b] $= [x_i, x_{i+1}]$

h $= \dfrac{b-a}{n}$

$x_i = a + i * h$

$x_{i+1} = x_i + h$

$y_{i+1} = y_i + Steigung$

0.3 Eulerverfahren

$y'(x) = f(x, y(x))$, $y_0 = y(a)$, $x_0 = a$

Algorithmus:

Geg AWP:

$y'(x) = f(x, y(x))$, $y_0 = y(a)$

Verfahren:

$x_{i+1} = x_i + h$

$y_{i+1} = y_i + h * f(x_i, y_i)$

0.4 Mittelpunktverfahren

Algorithmus:

Geg AWP:

$y'(x) = f(x, y(x))$, $y_0 = y(a)$

Verfahren:

$x_{h/2} = x_i + \frac{h}{2}$
$y_{h/2} = y_i + \frac{h}{2} * f(x_i, y_i)$
$x_{i+1} = x_i + h$
$y_{i+1} = y_i + h * f(x_{h/2}, y_{h/2})$

0.5 Das modifizierte Eulerverfahren

Algorithmus:
Geg AWP:
$y'(x) = f(x, y(x)),\ y_0 = y(a)$
Verfahren:
$x_{i+1} = x_i + h$
$yEuler_{i+1} = y_i + h * f(x_i, y_i)$
$k1 = f(x_i, y_i)$
$k2 = f(x_i, yEuler_{i+1})$
$y_{i+1} = y_i + h * (\frac{k1 + k2}{2})$

0.6 Fehlerordnung eines Verfahrens

Lokaler Fehler: Fehler nach einem Verfahren
$\varphi(x_i, h) := y(x_{i+1}) - y_{i+1}$
Konsistenzordung p falls:
$\varphi(x_i, h) \leq C * h^{p+1} \implies$ genügend kleine h und C>0

Globaler Fehler: Gesamtfehler also nach n-Iterationen
$y(x_n) - y_n$
Konvergenzordung p falls:
$|y(x_n) - y_n| \leq C * h^p \implies$ genügend kleine h und C>0

- Für die hier betrachteten Verfahren: Konsistenzordung = Konvergenzordung
- Verwendbar: Nur Verfahren mit Konvergenzordung p $\geq 1 \implies$ Globaler Fehler gegen 0.

Eulerverfahren: $p = 1$
Mittelpunktsregel: $p = 2$
Mod.Eulerverfahren: $p = 2$

0.7 Das klassische 4-Stufige Runge-Kutta Verfahren

Algorithmus:
Geg AWP:
$y'(x) = f(x, y(x)),\ y_0 = y(a)$
Verfahren:
$x_{i+1} = x_i + h$
$k1 = f(x_i, y_i)$
$k2 = f(x_i + \frac{h}{2}, y_{i+1} + \frac{h}{2})$
$k3 = f(x_i + \frac{h}{2}, y_{i+1} + \frac{h}{2} * k2)$
$k4 = f(x_i + h, y_{i+1} + h * k3)$
$y_{i+1} = y_i + h * \frac{1}{6}(k1 + 2k2 + 2k3 + k4)$

0.7.1 Das allgemeine s-stufige Runge-Kutta Verfahren

K$_n$ $= f(x_i + c_n * h, y_i + h * \sum_{m=1}^{n-1} a_{nm} k_m)$ n=1,...,s
$y_{i+1} = y_i + h * \sum_{n=1}^{s} b_n k_n$
s = Stufenzahl, a_{nm},b_n,c_n = Konstanten

c_1					
c_2	a_{21}				
c_3	a_{31}	a_{32}			
.	.	.			
.	.	.			
.	.	.			
c_n	a_{n1}	a_{n2}	...	$a_{n,n-1}$	
.	.	.	...	.	
.	.	.	...	.	
.	.	.	...	.	
c_s	a_{s1}	a_{s2}	...	$a_{s,s-1}$	
	b_1	b_2	...	b_{s-1}	b_s

Euler-Verfahren,s=1

0	
	1

Mittelpunkt-Verfahren,s=2

0		
0.5	0.5	
	0	1

Modifiziertes-Verfahren,s=2

0		
1	1	
	0.5	0.5

Klass.Runge-Kutta Verfahren, s=4

0				
0.5	0.5			
0.5	0	0.5		
1	0	0	1	
	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{6}$

```
def interpolate_runge_kutta_custom(f, x, h, y0):
    s = 4

    a = np.array([
        [0, 0, 0, 0],
        [1, 0, 0, 0],
        [1, 1, 0, 0],
        [0.75, 0.5, 0.75, 0]
    ], dtype=np.float64)
    b = np.array([0.1, 0.1, 0.4, 0.4], dtype=np.float64)
    c = np.array([0.75, 0.25, 0.75, 0.5], dtype=np.float64)
```

```
    y = np.full(x.shape[0], 0, dtype=np.float64)
    y[0] = y0

    for i in range(x.shape[0] - 1):
        k = np.full(s, 0, dtype=np.float64)

        for n in range(s):
            k[n] = f(x[i] + (c[n] * h), y[i] + h * np.sum([a[n][m] *
             k[m] for m in range(n - 1)]))

        y[i + 1] = y[i] + h * np.sum([b[n] * k[n] for n in range(s)])

    return y
```

0.8 Zurückführen DGL k-ter Ordnung auf k DGL 1.Ordnung

<u>Beispiel (3.Ordnung):</u>
$y''' + 5y'' + 8'y + 6y = 10e^{-x}$
<u>AWP:</u>
$y(0) = 2, y'(0) = y'(0) = 0$

<u>1.Schritt:</u>
Nach höchsten Ableitung auflösen:
$y''' = 10e^{-x} - 5y'' - 8y' - 6y$

<u>2.Schritt:</u>
Hilfsfunktionen bis (höchsten-1)Ableitung:
$z_1(x) = y(x)$
$z_2(x) = y'(x)$
$z_3(x) = y''(x)$

<u>3.Schritt:</u>
Hilfsfunktionen ableiten und in $z_3' = y'''$ einsetzen:
$z_1'(x) = y(x) = (z_2)$
$z_2'(x) = y''(x) = (z_3)$
$z_3'(x) = y'''(x)$
$z_3'(x) = 10e^{-x} - 5z_3 - 8z_2 - 6z_1$

<u>4.Schritt:</u>
DGL in vektorieller form:

$$z' = \begin{bmatrix} z_1' \\ z_2' \\ z_3' \end{bmatrix} = \begin{bmatrix} z_2 \\ z_3 \\ 10e^{-x} - 5z_3 - 8z_2 - 6z_1 \end{bmatrix} = f(x, z)$$
$$z(0) = \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} y(0) \\ y'(0) \\ y''(0) \end{bmatrix}$$

weil DGL 3.Ord, als LGL schreiben möglich: $z' = Az + b$

A= $\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -6 & -8 & -5 \end{bmatrix}$, b= $\begin{bmatrix} 0 \\ 0 \\ 10e^{-x} \end{bmatrix}$

Beispiel Eulerverfahren:

```
a = 0.
```

```python
b = 1.
h = 0.1
n = np.int((b-a)/h)
rows = 4

x = np.zeros(n+1)
z = np.zeros([rows,n+1])


x[0] = a
z[:,0] =np.array([0.,2.,0.,0.])

def f(x,z):
    return np.array([z[1], z[2], z[3], np.sin(x)+5-1.1*z[3]+0.1*z[2]+0.3*z[0]])

for i in range(x.shape[0]-1):
    x[i+1]=x[i]+h
    k1 = f(x[i], z[:,i])
    k2 = f(x[i] + (h / 2.0), z[:,i] + (h / 2.0) * k1)
    k3 = f(x[i] + (h / 2.0), z[:,i] + (h / 2.0) * k2)
    k4 = f(x[i] + h, z[:,i] + h * k3)
    z[:,i+1] = z[:,i] + h*(1/6)*(k1+2*k2+2*k3+k4)
```

0.9 Lösen eines Systems von k DGL 1.Ordnung

Hier $y^{(i)}$ = Vektor nach i-ten Iteration!!!

Rezept Lösungsverfahren:

Beispiel

$x_{i+1} = x_i + h$

$y^{(i+1)} = y^i + h * f(x_i, y^{(i)})$

$y''' = 10e^{-x} - 5y'' - 8y' - 6y$

System

$$z' = \begin{bmatrix} z_1' \\ z_2' \\ z_3' \end{bmatrix} = \begin{bmatrix} z_2 \\ z_3 \\ 10e^{-x} - 5z_3 - 8z_2 - 6z_1 \end{bmatrix} = f(x,z)$$

$$z(0) = \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} z_1^{(0)} \\ z_2^{(0)} \\ z_3^{(0)} \end{bmatrix}$$

i=0:

$$f(x_0, z^{(0)}) = \begin{bmatrix} z_2^{(0)} \\ z_3^{(0)} \\ 10e^{-x_0} - 5z_3^{(0)} - 8z_2^{(0)} - 6z_1^{(0)} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -2 \end{bmatrix}$$

$x_1 = x_0 + h = 0.5$

$$z^{(1)} = z^{(0)} + h * f(x_0, z^{(0)}) = \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix} + 0.5 \begin{bmatrix} 0 \\ 0 \\ -2 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ -1 \end{bmatrix}$$

i=1:

$$f(x_0, z^{(0)}) = \begin{bmatrix} z_2^{(1)} \\ z_3^{(1)} \\ 10e^{-x_0} - 5z_3^{(1)} - 8z_2^{(1)} - 6z_1^{(1)} \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \\ 0.9347 \end{bmatrix}$$

$x_2 = x_1 + h = 1$

$$z^{(2)} = z^{(1)} + h * f(x_1, z^{(1)}) = \begin{bmatrix} 2 \\ 0 \\ -1 \end{bmatrix} + 0.5 \begin{bmatrix} 0 \\ -1 \\ 0.9347 \end{bmatrix} = \begin{bmatrix} 2 \\ -0.5 \\ -1.4673 \end{bmatrix}$$

```
a = 0
b = 60
h = 0.1
n = int((b-a)/h)
c = 0.16
m = 1
l = 1.2
g = 9.81

rows = 2
phi0 = np.pi/2
x = np.zeros(n+1)
z = np.zeros([rows,n+1])
x[0] = a
z[:,0] =np.array([phi0,0],dtype=np.float64)


def f(x,z):
    return np.array([z[1],-((c/m)*z[1])-(g/l)*np.sin(z[0])])

for i in range(x.shape[0] - 1):
    x[i+1]=x[i]+h
    k1 = f(x[i], z[:,i])
    k2 = f(x[i] + (h / 2.0), z[:,i] + (h / 2.0) * k1)
    k3 = f(x[i] + (h / 2.0), z[:,i] + (h / 2.0) * k2)
    k4 = f(x[i] + h, z[:,i] + h * k3)

    z[:,i+1] = z[:,i] + h * (1 / 6.0) * (k1 + 2 * k2 + 2 * k3 + k4)
```