1) Explizites Eulerverfahren

Analog zum Jupyter Notebook der Vorlesung implementiert. Anstelle i wurde y verwendet, und x anstelle t.

```python
def explizitEuler(xend, h, y0, f):
    x = [0.]
    y = [y0]
    xalt = 0
    yalt = y0

    while x[-1] < xend-h/2:
        # explizites Eulerverfahren
        yneu = yalt + h*f(xalt, yalt)
        xneu = xalt + h

        # Speichern des Resultats
        y.append(yneu)
        x.append(xneu)

        yalt = yneu
        xalt = xneu
    return np.array(x), np.array(y)
```

2) Mit Modellproblem testen

AWP
$$y'(x) = -4y(x) \qquad x \in [0,2]$$
$$y(0) = 1$$

Analytische Lösung bestimmen mit Formel aus AN2/AN3:

$$y(x) = e^{-a(x-x_0)} \cdot y_0 + e^{-ax} \int_{x_0}^{x} e^{at}\, g(t)\, dt \qquad \text{für AWP mit } y(x_0) = y_0$$

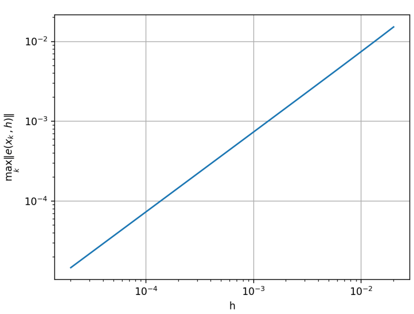
$$y(x) = e^{-ax} + \cancel{e^{-ax} \int_{0}^{x} e^{at} \cdot 0\, dt}^{\;0}$$

$$y(x) = e^{-4x}$$

Berechnung und Visualisierung des absoluten Fehlers wird in 5) beschrieben.

3) Kontrolle der Konvergenzordnung

Mittels Code aus dem Praktikumsbeschrieb



4) Implementation des impliziten Eulerverfahren

Analog zum Jupyter Notebook der Vorlesung und dem vorgegebenen Frame aus dem Praktikumsbeschrieb implementiert. Unterschied zur Version aus der Vorlesung: Mittels Newton-Verfahren (aus SW07 bekannt) wird die Nullstelle der generischen Abbildung $G(s) = s - y_k - h \cdot f(x_{k+1}, s)$ gesucht. Sowohl die generische Abbildungsfunktion $G(s)$ als auch deren partielle Ableitung nach s werden in der Funktion definiert.

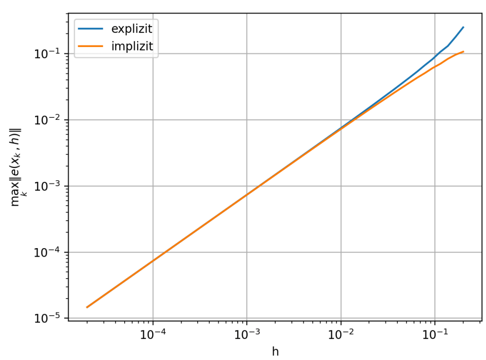
$$\partial_s G(s) = 1 - h \cdot \partial_s f(x_{k+1}, s)$$

```python
def implizitEuler(xend, h, y0, f, df):
    x = [0.]
    y = [y0]

    # Verfahrensfunktion für implizit Euler
    def G(s, xk, yk):
        return s - yk - h * f(xk + h, s)

    # Partielle Ableitung nach s der Verfahrensfunktion
    def dG(s, xk, yk):
        return 1 - h * df(xk + h, s)

    def newton(s, xk, yk, tol=1e-12, maxIter=20):
        k = 0
        delta = 10 * tol
        while np.abs(delta) > tol and k < maxIter:
            delta = G(s, xk, yk) / dG(s, xk, yk)
            s -= delta
            k += 1
        return s

    while x[-1] < xend - h / 2:
        y.append(newton(y[-1], x[-1], y[-1], tol=1e-12, maxIter=20))
        x.append(x[-1] + h)

    return np.array(x), np.array(y)
```

5) Kontrolle der Konvergenzordnung des impliziten Eulerverfahrens + Visualisierung absoluter Fehler

Code aus 3) so angepasst, dass sowohl implizite als auch explizite Konvergenzordnung im gleichen Diagramm dargestellt werden. Es ist zu erkennen, dass beide Verfahren eine Abweichung aufzeigen sobald das h zu gross gewählt wird.

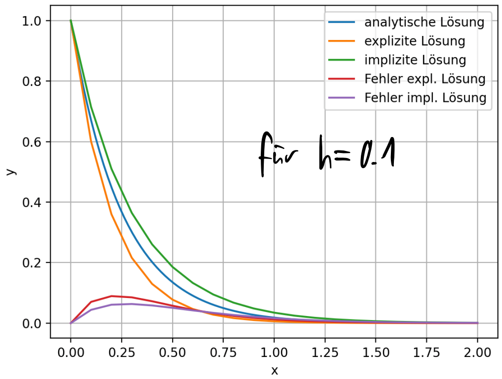
```python
# ----------- Kontrolle der Konvergenzordnung -----------------------------------
n = 10**np.linspace(1,5)
hs = 2/n
err_exp = []
err_imp = []
for h in hs:
    xe, ye = explizitEuler(2, h, 1, f)
    err_exp.append(np.linalg.norm(ye - ya(xe), np.inf))  # ya(xe) ist die exakte Lösung am Punkt xe

    xi, yi = implizitEuler(2,h,1,f,df)
    err_imp.append(np.linalg.norm(yi-ya(xi),np.inf)) # ya(xi) ist die exakte Lösung am Punkt xi

plt.loglog(hs,err_exp,'-', label='explizit')
plt.loglog(hs,err_imp,'-', label='implizit')
plt.xlabel('h')
plt.ylabel(r'$\max_k \|e(x_k,h)\|$')
plt.legend()
plt.grid()
plt.show()
```

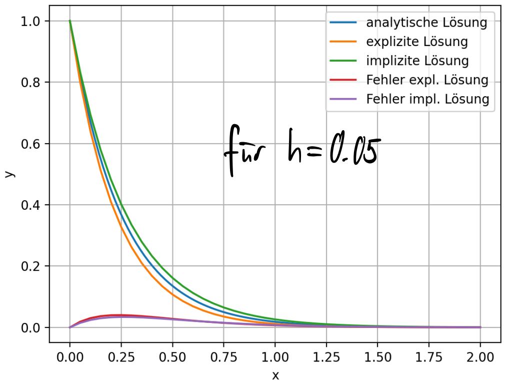


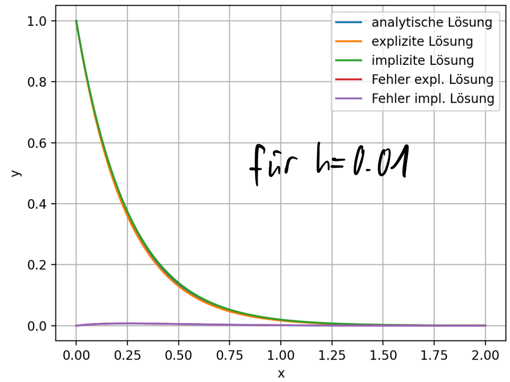
Um den absoluten Fehler zu visualisieren wurden die beiden Eulerverfahren für mehrere Werte von h durchgeführt, und anschliessend der Abstand zur analytisch berechneten Lösung ermittelt. Die absoluten Fehler sind gemeinsam mit der analytischen, impliziten und expliziten Lösung dargestellt. Es ist gut ersichtlich, dass der absolute Fehler der impliziten Lösung bei etwas grösseren h (h=0.1) kleiner ist als jener der impliziten Lösung.

Code:

```python
# ------------ Berechnung des absoluten Fehlers --------------------
xp = np.linspace(0,2,100)
xe, ye = explizitEuler(2, 0.1, 1, f)
xi, yi = implizitEuler(2, 0.1, 1, f, df)

plt.figure('absoluter Fehler')
plt.plot(xp, ya(xp),'-', label='analytische Lösung')
plt.plot(xe, ye,'-', label='explizite Lösung')
plt.plot(xi, yi,'-', label='implizite Lösung')
plt.plot(xe, np.abs(ye - ya(xe)),'-', label='Fehler expl. Lösung')
plt.plot(xe, np.abs(yi - ya(xi)),'-', label='Fehler impl. Lösung')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.grid()
plt.show()
```







6) AWP mit den implementierten Verfahren berechnen

AWP
$$\begin{cases} y'(x) = -x^2 \cdot \dfrac{1}{y(x)} \\ y(0) = -4 \end{cases}$$

Analytische Lösung des AWP's bestimmen: Die DGL ist separierbar, das heisst sie kann in die Form $y' = f(x)\,g(y)$ zerlegt werden, mit $f(x) = -x^2$ und $g(y(x)) = \frac{1}{y(x)}$.

$$\frac{dy}{dx} = -x^2 \cdot \frac{1}{y}$$

$$y\,dy = -x^2\,dx$$

$$\int_{-4}^{y} y\,dy = -\int_{0}^{x} x^2\,dx$$

$$\frac{1}{2}y^2 \Big|_{-4}^{y} = -\frac{1}{3}x^3 \Big|_{0}^{x}$$

$$\frac{1}{2}y^2 - \frac{1}{2}\cdot 16 = -\frac{1}{3}x^3 - 0$$

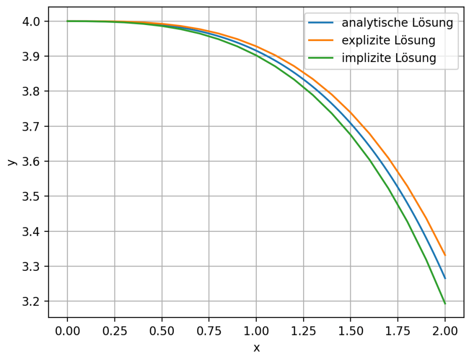
$$y(x) = -\sqrt{16 - \frac{2}{3}x^3}$$

Kontrolle: $y(2) = -\sqrt{16 - \frac{2}{3}2^3} = -4\sqrt{\frac{2}{3}}$ ✓

AWP mit beiden Verfahren berechnen und darstellen:

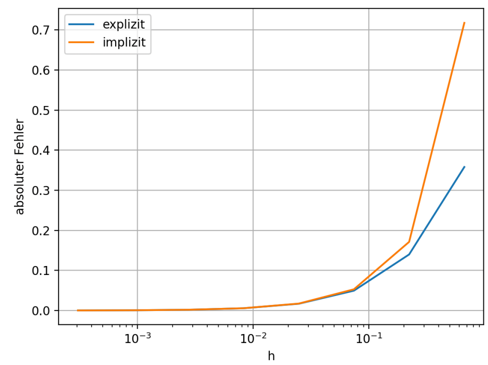
```python
# ------------ AWP mit beiden Verfahren berechnen und darstellen ----
xe, ye = explizitEuler(2, 0.1, 4, f)
xi, yi = implizitEuler(2, 0.1, 4, f, df)
xp = np.linspace(0,2,100)
plt.figure('Lösung des AWP')
plt.plot(xp, ya(xp),'-', label='analytische Lösung')
plt.plot(xe, ye,'-', label='explizite Lösung')
plt.plot(xi, yi,'-', label='implizite Lösung')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.grid()
plt.show()
```

(hier mit h=0.1)



absoluten Fehler für verschiedene Schrittweiten berechnen und darstellen:

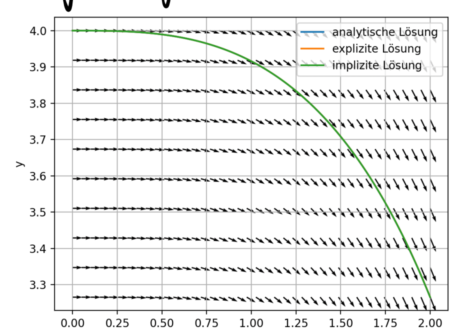
```python
# ------------ absoluten Fehler für verschiedene Schrittweiten berechnen und darstellen ------------
hs = []
for j in np.linspace(1,8,8):
    hs.append(2/(3**j))        # die einzelnen h's berechnen, wie im Praktikumsbeschrieb vorgegeben

err_exp = []
err_imp = []
for h in hs:
    xe, ye = explizitEuler(2, h, 4, f)
    err_exp.append(np.linalg.norm(ye - ya(xe), np.inf))  # ya(xe) ist die exakte Lösung am Punkt xe
    xi, yi = implizitEuler(2, h, 4, f, df)
    err_imp.append(np.linalg.norm(yi - ya(xi), np.inf))  # ya(xi) ist die exakte Lösung am Punkt xi

plt.figure('Konvergenzordnung')
plt.semilogx(hs,err_exp,'-', label='explizit')
plt.semilogx(hs,err_imp,'-', label='implizit')
plt.xlabel('h')
plt.ylabel('absoluter Fehler')
plt.legend()
plt.grid()
plt.show()
```



Visualisierung der Lösung im Richtungsfeld der Differentialgleichung:

Hierfür wurden die beiden unten markierten Zeilen aus dem Jupyter Notebook der Vorlesung übernommen, angepasst und in den Code zum plotten der Lösungen eingebettet.

```python
# ------------ Lösung im Richtungsfeld visualisieren, für N=3^8 --------------
h = 2/(3**8)
xe, ye = explizitEuler(2, h, 4, f)
xi, yi = implizitEuler(2, h, 4, f, df)
xp = np.linspace(0,2,100)
plt.figure('Lösung im Richtungsfeld')
plt.plot(xp, ya(xp),'-', label='analytische Lösung')
plt.plot(xe, ye,'-', label='explizite Lösung')
plt.plot(xi, yi,'-', label='implizite Lösung')
xq,yq = np.meshgrid(np.linspace(0,2,int(2/.05)),np.linspace(np.min(ya(xp)),np.max(ya(xp)),10))
plt.quiver(xq,yq,np.ones_like(xq),f(xq,yq),angles='xy')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.grid()
plt.show()
```



7) Unterschied zwischen explizitem und implizitem Verfahren

Der Unterschied besteht darin, dass sich die Steigung beim expliziten Verfahren am aktuellen Punkt orientiert, beim impliziten Verfahren jedoch jeweils am nächsten Punkt. Mit dem Richtungsfeld von oben und Schrittweite $h = 0.5$ lässt sich dies sehr schön erkennen: