Lösung mit Determinanten (1)

- Im ersten Lösungsansatz werden Determinanten verwendet:

```
det = np.linalg.det(x)
print("determinant:")
print(det)
```

  Die Funktion **det** im Submodul linalg berechnet die Determinante der Matrix.

- Falls die Determinante ungleich 0 ist, gibt es eine eindeutige Lösung des Gleichungssystems. Wir berechnen die Lösung mit Hilfe der Cramerschen Regel.
- Dazu brauchen wir zuerst einen Container für die zu berechnende Lösung, d.h. einen Lösungsvektor mit beliebigen Inhalt. Diesen erhalten wir durch die Funktion **zeros**:

```
s = np.zeros(3)
```

- Natürlich könnten wir s auch explizit angeben:

```
s = np.array([0.0, 0.0, 0.0])
```

Lösung mit Determinanten (2)

- Es geht auch durch Aufzählung:

```
s = np.arange(3, step=1.0).reshape(3)
```

  Die Funktion **arange** zählt die Zahlen 0.0, 1.0, 2.0 auf und die Funktion reshape erstellt daraus ein Array.
- Da die Werte unwichtig sind, kann man auch einfach einen anderen Vektor kopieren:

```
s = y.copy()
```

- Um einen Vektor oder eine Matrix herzustellen, gibt es in numpy also folgende Möglichkeiten:
  - Explizite Angabe
  - Aufzählung
  - zeros oder die Funktion ones
  - Kopieren eines bestehenden Vektors oder einer bestehenden Matrix

Lösung mit Determinanten (3)

- Mit der Cramerschen Regel wird die Lösung des Gleichungssystems folgendermaßen berechnet:

$$\begin{bmatrix} \text{det(m1) / det(x)} \\ \text{det(m2) / det(x)} \\ \text{det(m3) / det(x)} \end{bmatrix}$$

  wobei mi die Matrix ist, die aus x durch Ersetzung der i-ten Spalte durch y entsteht.

- Wir berechnen die Lösung in einer Schleife:

```
print("solution:")
for i in range(3):
    mi = x.copy()                        # x kopieren
    mi[:, i] = y                         # i-te Spalte durch y ersetzen
    s[i] = np.linalg.det(mi) / det       # i-te Lösung berechnen
print(s)
```

Lösung mit Determinanten (4)

- Um die Ersetzung der i-ten Spalte durch y zu verstehen, betrachten wir den Zugriff auf Vektor- und Matrixelemente. Beispiele:

```
print(y)                # Ausgabe: [-2 -8  5]
print(y[0])             # Ausgabe: -2
print(y[1:3])           # Ausgabe: [-8  5]
print(y[ : ])           # Ausgabe: [-2 -8  5]

print(x)                # Ausgabe: [[ 1.     1.     1.   ]
                                    [27.     9.     3.   ]
                                    [42.875 12.25   3.5  ]]
print(x[0, 0])          # Ausgabe: 1.0
print(x[2, 2])          # Ausgabe: 3.5
print(x[ : , 0])        # Ausgabe: [ 1.    27.    42.875]  erste Spalte
print(x[2 , : ])        # Ausgabe: [42.875 12.25   3.5  ] letzte Zeile
print(x[0:2 , 0:2])     # Ausgabe: [[ 1.  1.]
                                    [27.  9.]]
```

Lösung mit Determinanten (5)

- Um das berechnete Polynom zu visualisieren verwenden wir matplotlib. Auch dieses Modul muss vor der Verwendung installiert

  ```
  pip install matplotlib
  ```

  und importiert werden:

  ```
  from matplotlib import pyplot as plt
  ```

- Zuerst setzen wir die Beschriftungen des Plots und Achsen mit den Funktionen **title**, **xlabel**, **ylabel**:

  ```
  plt.title("Raumsonde Cobra")
  plt.xlabel("x")
  plt.ylabel("y")
  ```

- Der zu zeichnende Graph besteht aus Punkten, die zu einer Linie verbunden werden. Die x-Werte 0.0, 0.1, 0.2, …, 3.9 der Punkte werden folgendermaßen angegeben:

  ```
  x_plot = np.arange(4, step=0.1)
  ```

Lösung mit Determinanten (6)

- Die y-Werte sind

```
y_plot = s[0] * x_plot**3 + s[1] * x_plot**2 + s[2] * x_plot
```

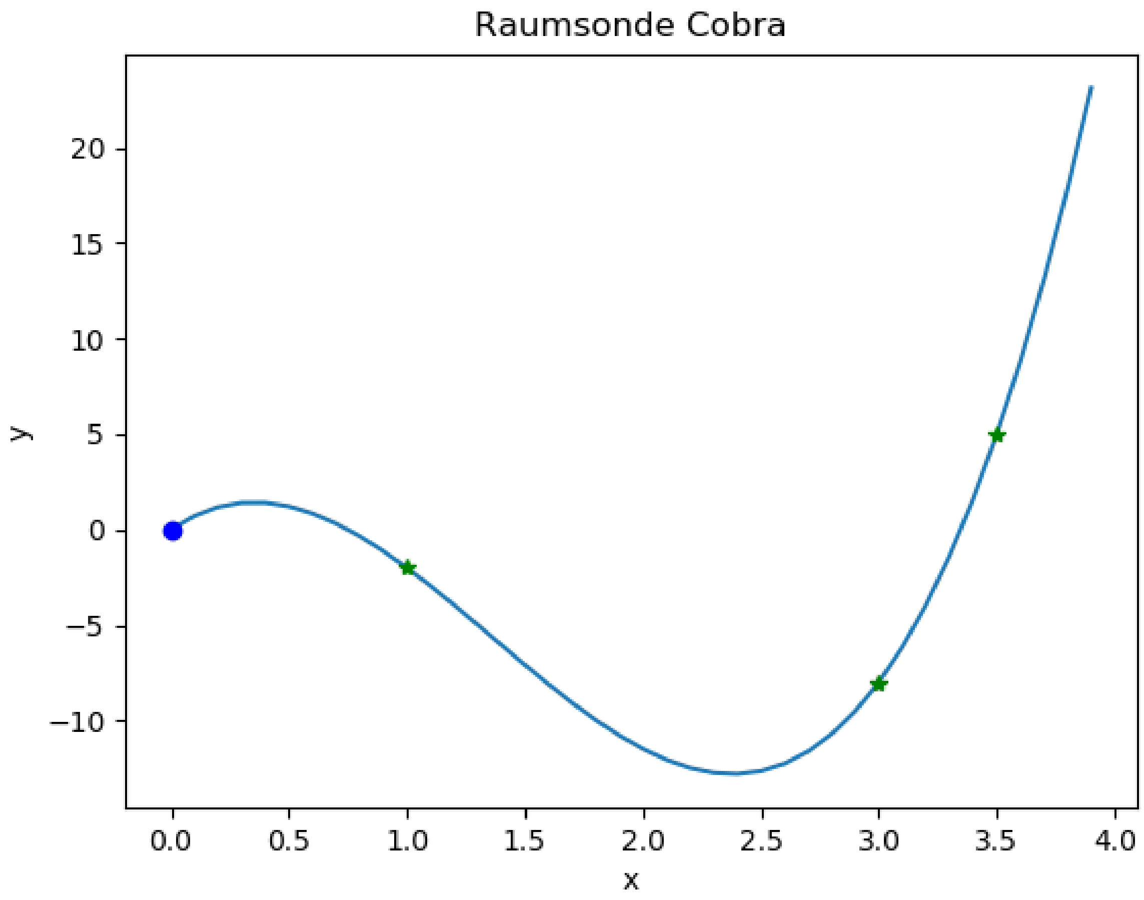
- Die Funktion **plot** zeichnet die Punkte. Es wird das x- und das y-Array angegeben:

```
plt.plot(x_plot, y_plot)
```

- Zur Kontrolle geben wir die Stützpunkte aus:

```
plt.plot([x1, x2, x3], [y1, y2, y3], "*g")
plt.plot([0], [0], "ob")
plt.show()
```

Zum Schluss die Funktion **show** nicht vergessen.



Lösung mit Determinanten (7)

- Die Funktion plot kann folgenermaßen verwendet werden:

  ```
  plt.plot(x, y, fmt)
  ```

  x und y sind die Koordinaten der zu zeichnenden Punkte. x kann auch weggelassen werden. Der Default ist das Indexarray 0, 1, 2, …, n – 1 wobei n die Länge von y ist.

  fmt ist ein Formatstring. Beispiele:

  ```
  "*g“            # grüne Sterne
  "ob“            # blaue Kreise
  ```

  Weitere Formatierungen und Farben: - -- -. : , o v ^ < > 1 2 3 4 s p * h H + x D | _   b g r c m y k w
  Wird der Formatstring nicht angegeben, wird der Default verwendet.

- plot kann auch öfter aufgerufen werden, um mehrere Plots in ein Diagramm zu zeichnen.
- Näher wollen wir auf matplotlib nicht eingehen. Probieren Sie es selber aus!

Lösung mit Determinanten (8)

Aufgaben

1. Auf den vorangegangenen Folien haben wir die Lösung des linearen Gleichungssystem als Python-Skript behandelt. In dieser Aufgabe soll der Lösungsalgorithmus als Funktion geschrieben werden.

def cramer(x, y):

Parameter:	array x : Koeffizientenmatrix
	array y : Konstantenvektor
Rückgabewert:	array : Lösungsvektor

Zur Berechnung der Lösung können Sie die folgenden Anweisungen verwenden:

```
det = np.linalg.det(x)
s = np.zeros(3)
for i in range(3):
    mi = x.copy()
    mi[:, i] = y
    s[i] = np.linalg.det(mi) / det
```

Lösung mit Determinanten (9)

Aufgaben

Die Funktion soll aber nicht nur für drei Gleichungen sondern für eine beliebige Anzahl funktionieren. Das heißt, Sie müssen überall, wo die Konstante 3 verwendet wird, die Anzahl der Gleichungen einsetzen. Diese Anzahl bekommen Sie z.B. durch y.size.

Rufen Sie die Funktion mit der Matrix x und dem Vektor y wie auf den vorangegangenen Folien auf. Plotten Sie das Ergebnispolynom.

2. Ergänzen Sie die Funktion um die Prüfung, ob die Anzahl der Zeilen von x und die Anzahl der Spalten von x und die Anzahl der Elemente von y gleich sind. Recherchieren Sie dazu im Netz, wie man die Anzahl der Zeilen und die Anzahl der Spalten eines Arrays ermittelt. Außerdem soll überprüft werden, ob die Determinante nicht 0 ist. Im negativen Fall soll jeweils np.array([None]) zurückgegeben werden.
3. Fügen Sie einen beliebigen vierten Planeten hinzu.