Modellierung eines Schiffskurses

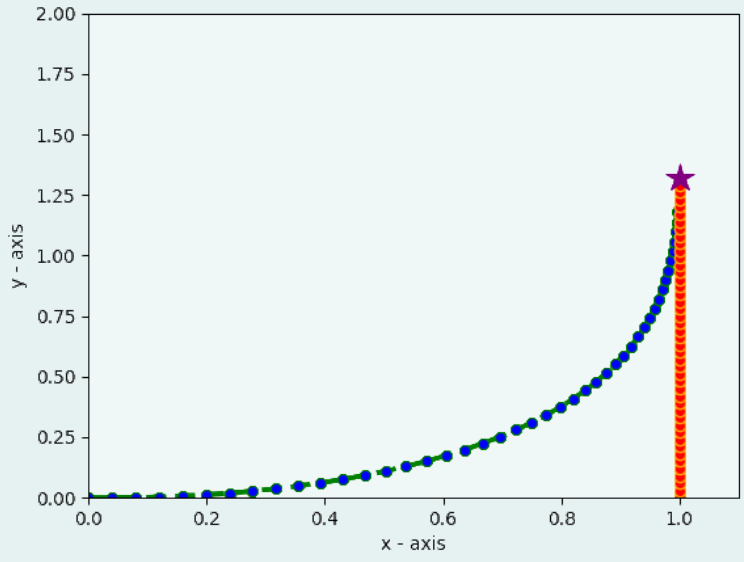
Dennis L. Buchholz

April 13, 2023

Einleitung

Es soll ein Simulationsprogramm in den Programmiersprachen C und Python implementiert werden, welches ein Szenario modelliert, in dem ein Handelsschiff mit konstanter Geschwindigkeit segelt und von einem Piratenschiff verfolgt wird. Das Ziel des Programms ist es, den Zeitpunkt (th0) sowie die kartesische Y-Koordinate (yh0) des Treffpunkts der beiden Schiffe zu berechnen.

Beispielgrafik



Graphische Ausgabe des beiliegenden Programms (1)

Technische Hinweise

Für die Kompilierung und das Testen des C-Quelltexts wurde der Compiler **gcc** in Version 9.2.0, GNU **make** in Version 3.82.90 und GNU **ld** (binutils 2.32) aus MinGW genutzt. Für den Python-Quelltext wurde **cpython** 3.10.9 mit den Bibliotheken `matplotlib` und `numpy` verwendet.
Zur Erstellung der Dokumentation kam **texify** von MiKTeX in Version 22.8.28 zum Einsatz.

1 Allgemeine Struktur des Programms

In diesem Abschnitt werden die verwendeten Datenstrukturen und mathematischen Konzepte beschrieben.

1.1 Datenstrukturen

Für das Programm werden folgende zwei Datenstrukturen definiert:

```
@dataclass
class Point:
    x: float
    y: float
```

Eine Datenstruktur, welche einen Punkt auf einem kartesischen Graphen mit einer X- und Y-Koordinate repräsentiert.

```
@dataclass
class Result:
    th0: float
    yh0: float
```

Eine Datenstruktur, welche die geforderten Werte th0 und yh0 zusammenfasst.

1.2 Mathematische Theorie

Die Berechnung in den beiliegenden Programmen basiert auf der in der Aufgabe gegebenen Tabelle:

k	t	x_h	y_h	x	y	$\tan\phi$	Δx	Δy
k_0	$k_0\Delta t$	1	$k_0 n\Delta t$	$x^{(k_0-1)} + \Delta x^{(k_0-1)}$	$y^{(k_0-1)} + \Delta y^{(k_0-1)}$	$\frac{y_h^{(k_0)} - y^{(k_0)}}{1-x^{(k_0)}}$	$\Delta t \cos\phi^{(k_0)}$	$\Delta t \sin\phi^{(k_0)}$

Die Iteration kann beendet werden wenn $|x_h - x| < \epsilon$ und $|y_h - y| < \epsilon$

Der Wert ε ist der Abstand zwischen den Schiffen, der mindestens erreicht werden muss, um th0 und yh0 zu bestimmen. Δt repräsentiert jeweils einen Zeitabschnitt in der Simulation. Beide Werte sollten jeweils klein gewählt werden, um th0 und yh0 möglichst genau zu bestimmen. Um eine mathematisch exakte Bestimmung zu erzielen, müsste der Grenzwert beider Werte ermittelt werden:

Mathematisches Beispiel 1.1

$$\lim_{\epsilon \to 0}$$

$$\lim_{\Delta t \to 0}$$

Um den Wert φ auszurechnen (im folgenden Programmcode *distance* genannt), wird die Arkustangensfunktion verwendet. Die Arkustangensfunktion, dargestellt durch atan(x) oder $\tan^{-1}(x)$, ist eine Funktion, die als Eingabe eine reelle Zahl x annimmt und ihren Winkel in Bogenmaß zurückgibt, dessen Tangens gleich x ist. Mit anderen Worten, wenn $y = \text{atan}(x)$, dann gilt $\tan(y) = x$, wobei x eine reelle Zahl und y der entsprechende Winkel in Bogenmaß ist.
Der Definitionsbereich der Funktion atan umfasst alle reellen Zahlen, aber ihr Wertebereich ist auf das Intervall $\left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$, oder etwa $(-1.57, 1.57)$ in Bogenmaß, begrenzt. Dies bedeutet, dass der Ausgabewert der Funktion atan unabhängig vom Eingabewert immer eine Zahl zwischen $-\frac{\pi}{2}$ und $\frac{\pi}{2}$ ist. Daher wird im Falle eines Fehlers, bei dem der Divisor 0 ist, die Rückgabe der Funktion atan als $\frac{\pi}{2}$ definiert.

1.3 Implementation der Rechenschritte

Die Berechnung des Schiffskurses wird folgendermaßen in Python implementiert:

```
@tail_recursive
def approx_p_course(k, t: float, delta_t, pos_h: Point, pos_p: Point, angle, delta_x, delta_y, epsilon
    ):
```

N wird als die Konstante 3/4 definiert. *delta_t* und *epsilon* werden als feste, aber beliebig kleine, reelle Zahlen angenommen. Daraus ergeben sich folgende Rechenschritte analog zu den Rechenschritten der Tabelle:

```
pos_h.y = k * N * delta_t
```

Die Y-Koordinate des Handelsschiffes ist das Produkt aus der Multiplikation, des Iterators k, der Konstante N und der Variable *delta_t*.

```
t = k * delta_t
```

Die aktuelle Zeit berechnet sich aus dem Produkt des Iterators und der Variable *delta_t*.

```
try:
    distance = (pos_h.y - pos_p.y) / (1 - pos_p
    .x)
except ZeroDivisionError:
    distance = numpy.pi / 2
```

Es sei *distance* ein Quotient, bei welchem der Dividend die Differenz der Y-Koordinaten der beiden Schiffe ist und der Divisor die Differenz der beiden X-Koordinaten. Da die X-Koordinate des Handelsschiffes konstant ist, bedarf sie keiner Variable.
Es ist zu beachten, dass in diesem Rechenschritt ein Fehler auftreten kann, wenn der Divisor 0 wird. Dies geschieht, wenn die X-Koordinate des Piratenschiffes den Wert 1 annimmt. Um diesen Fehler zu behandeln, wird in diesem Fall *distance* als pi/2 definiert.

```
pos_p.x += delta_x
pos_p.y += delta_y

if pos_p.x > pos_h.x + 1:
    print("Distance is never below given
    epsilon threshold!")
    return None
```

Die Koordinaten des Piratenschiffes werden jeweils um das *delta_x* und *delta_y* der aktuellen Iteration erhöht. Des Weiteren wird der Fall überprüft, ob die X-Koordinate des Piratenschiffes bereits größer als die X-Koordinate + 1 sei. Hiermit wird überprüft, ob das Piratenschiff das Handelsschiff bereits überholt habe. Ist dies der Fall, so ist niemals der zu erreichende Schwellwert *epsilon* erreicht und die Simulation muss mit einer Fehlermeldung beendet werden. Dieser Fall muss überprüft werden, da die Abbruchbedingung (*epsilon*) in diesem Fall sonst nie erreicht werde.

```
angle = numpy.arctan(distance)
delta_x = delta_t * numpy.cos(angle)
delta_y = delta_t * numpy.sin(angle)
```

Die Arkurstangensfunktion wird verwendet, um den Winkel (*angle*) zwischen den beiden Schiffen zu berechnen. Dabei wird die Variable *distance* als Parameter in die Funktion eingesetzt.
Um den Wert der Variable *delta_x* zu berechnen, wird das Produkt aus *delta_t* und der Kosinusfunktion des Winkels (*angle*) verwendet. Hierfür wird der Winkel als Parameter in die Kosinusfunktion eingesetzt. Analog dazu wird der Wert der Variable berechnet, *delta_x* berechnet, jedoch wird die Sinusfunktion anstelle der Kosinusfunktion verwendet.

```
if (abs(pos_h.x - pos_p.x) < epsilon) and (abs(
    pos_h.y - pos_p.y) < epsilon):
    result.th0 = t
    result.yh0 = pos_h.y
else:
    approx_p_course(k+1, t, delta_t, pos_h,
    pos_p, angle, delta_x, delta_y, epsilon)
```

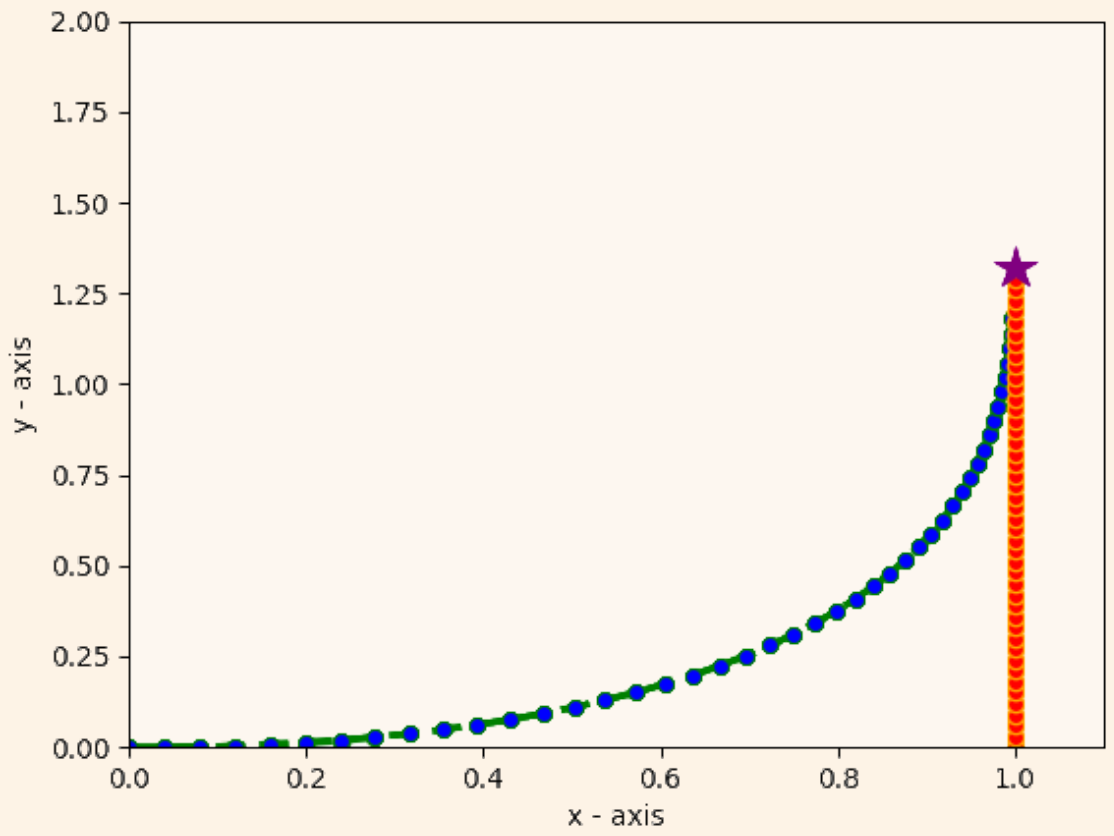
Ist sowohl der Betrag der Differenz der X-Koordinaten kleiner als der Schwellwert *epsilon*, als auch der Betrag der Differenz der Y-Koordinaten kleiner als der Schwellwert *epsilon*, dann sei die Abbruchbedingung erreicht, und die gesuchten Werte *th0* und *yh0* ergeben sich aus der Variable t und der aktuellen Y-Koordinate des Handelsschiffes.

2 Auswertung der Ergebnisse

Beispiel 2.1. Rekursive Python-Variante

```
if __name__ == '__main__':
    position_merchants = Point(1, 0)
    position_pirates = Point(0, 0)
    t_delta_start = 0.04
    epsi = 0.15

    approx_p_course(0, 0, t_delta_start, position_merchants, position_pirates, 0, 0, 0, epsi)
    print(result)
    plt.plot(pirate_plot_x, pirate_plot_y, color='green', linestyle='dashed', linewidth=3,
             marker='.', markerfacecolor='blue', markersize=12)

    merchant_plot_x = merchant_plot_y.copy()
    merchant_plot_x.fill(1)
    plt.plot(merchant_plot_x, merchant_plot_y, color='orange', linestyle='dashed', linewidth=3,
             marker='.', markerfacecolor='red', markersize=12)                    # plot merchant ship

    plt.plot([1], [result.yh0], label="stars", color="purple", markersize=16, marker="*") # plot the position of yh0

    plt.ylim(0, 2)                  # y-axis range
    plt.xlim(0, 1.1)                # x-axis range
    plt.xlabel('x - axis')          # naming the x axis
    plt.ylabel('y - axis')          # naming the y axis
    plt.title('Ships ships ships!') # giving a title to my graph
    plt.show()                      # function to show the plot
```

Beispieleingabe in das beiliegende Programm, $delta\_t = 0.04$, $epsilon = 0.15$



Graphische Ausgabe (th0=1.76, yh0=1.32)

Bemerkung

Im vorliegenden Python-Code zur Berechnung des Schiffskurses wird eine endrekursivse Funktion verwendet, anstelle einer iterativen Funktion. Obwohl endrekursive Funktionen oft als effizienter gelten, führt die Verwendung in diesem Fall zu Einschränkungen im Definitionsbereich der Funktion. Insbesondere lassen sich die Parameter *epsilon* und *delta_t* nicht hinreichend klein wählen, da ab einer gewissen Rekursionstiefe die Grenze des Stack-Speichers erreicht wird.
Daher ist es ratsam, für eine Python-Implementierung eine iterative Variante zu verwenden um Seiteneffekte zu vermeiden und ein möglichst genaues Ergebnis zu erhalten. Die C-Implementierung hingegen weist diesen Seiteneffekt nicht auf, da der C Compiler die Endrekursion optimiert.

Beispiel 2.2. Rekursive C-Variante (genaueres Ergebnis)

```c
int main (int argc, char** argv) {
    double delta_t = 0.000001;
    double epsilon = 0.000005;

    Point position_merchants = {.x = 1, .y = 0};
    Point position_pirates = {.x = 0, .y = 0};

    approx_course( k: 0, t: 0, delta_t, ph: position_merchants, pp: position_pirates, angle: 0, delta_x: 0, delta_y: 0, epsilon);
    return 0;
}
```

```
approx_course
Terminal:  Local  +
th0: 2.285696, yh0: 1.714272
```

Beispieleingabe in das beiliegende Programm ($delta\_t = 0.000001$, $epsilon = 0.000005$)

Mit der C-Implementierung wird ein hinreichend genaues Ergebnis erzielt.