Projekt: Numerisches Lösen

Bisektionsverfahren

Einleitung

Numerische Verfahren können auch Gleichungen und Gleichungssysteme lösen, für die keine analytischen Lösungen bekannt sind. Im Regelfall liefern diese Verfahren aber nur Näherungswerte, die aber beliebig genau berechnet werden können.

Die meisten numerischen Verfahren arbeiten **iterativ**. Das heißt, die Lösung wird schrittweise bis zur gewünschten Genauigkeit entwickelt. Praktisch erfolgt dies so, dass im ersten Schritt eine grobe Näherung bestimmt wird und durch geeignete mehrfache Anwendung des Verfahrens die Lösung verbessert wird.

Ein Tipp zur praktischen Arbeit, der auch in anderen Bereichen Gültigkeit hat: Ein neues Verfahren zuerst einmal von Hand an einem überschaubaren Kleinbeispiel nachzuvollziehen, um die Arbeitsweise zu verstehen.

Bestimmung von Nullstellen

Das klassische Verfahren zur Lösung von Gleichungen ist die Methode der Bestimmung der Nullstellen. Soll beispielsweise die Gleichung

$$x^2 + \sin x = 10$$

gelöst werden, so schreibt man diese um zu

$$f(x) = x^2 + \sin x - 10$$

und sucht die Nullstelle(n) dieser Funktion $f(x) = 0$.

Es existieren hunderte von numerischen Verfahren die Lösungen anbieten. Viele sind sehr speziell und nur geeignet um ganz bestimmte Funktionstypen zu bearbeiten.

Die beiden bekanntesten universell einsetzbaren numerischen Methoden sind:

- Methode der Intervallhalbierung
- Nullstellensuche nach Newton (Newton-Raphson Verfahren)

Diese Verfahren bestimmen beliebig genau eine reelle Nullstelle für eine gegebene Funktion mit Hilfe eines Iterationsverfahrens. Sie eignen sich praktisch für alle Klassen von Funktionen (Polynome, rationale, trigonometrische, hyperbolische Funktionen, etc.).

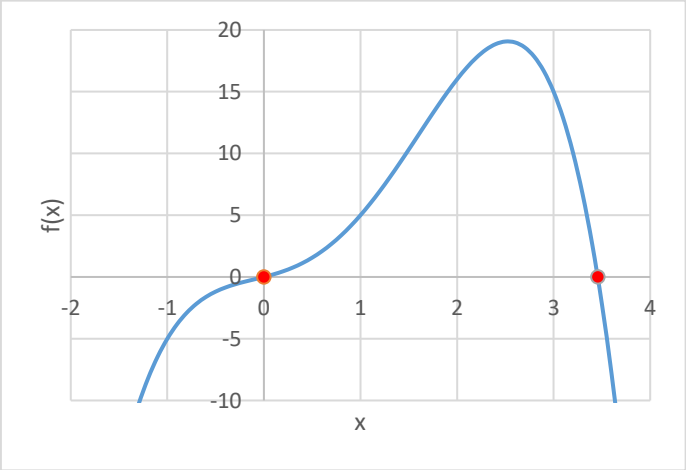
Iterative Lösungsverfahren liefern keine exakte Lösung der Gleichung. Sie liefern einen beliebig genauen Näherungswert. Die Genauigkeit kann prinzipiell bestimmt werden. Je genauer die gewünschte Lösung, desto größer der Rechenaufwand.

Bei den oben erwähnten numerischen Verfahren erfolgt die Suche von einem Startwert (= sogenanntes **Anfangswertproblem**) aus mit einem systematischen Iterationsverfahren, bis eine Lösung gefunden wird. Ein Problem ist bei allen Verfahren, dass zwar eine Lösung gefunden werden kann, diese aber nicht die einzige Lösung sein muss. So stellt sich die Frage wie viele Lösungen überhaupt existieren, oder falls keine Lösung gefunden wird, ob das System tatsächlich keine Lösung besitzt.

Die Beurteilung der Menge der Lösungen kann sicher aufgrund der Funktionseigenschaften teilweise beantwortet werden. Der einfachste Weg ist sicher das Aufzeichnen des **Graphen der Funktion**. Man kann damit aus dem Funktionsverlauf leicht bestimmen wo die reellen Lösungen zu vermuten sind und wie groß die Anzahl ist. Hierzu eignen sich heute vor allem die Mathematikpakete wie z.B. Mathematica (bzw. Wolfram-Alpha).

Beispiel

Polynom $P_4(x) = 2x + x^2 + 3x^3 - x^4$



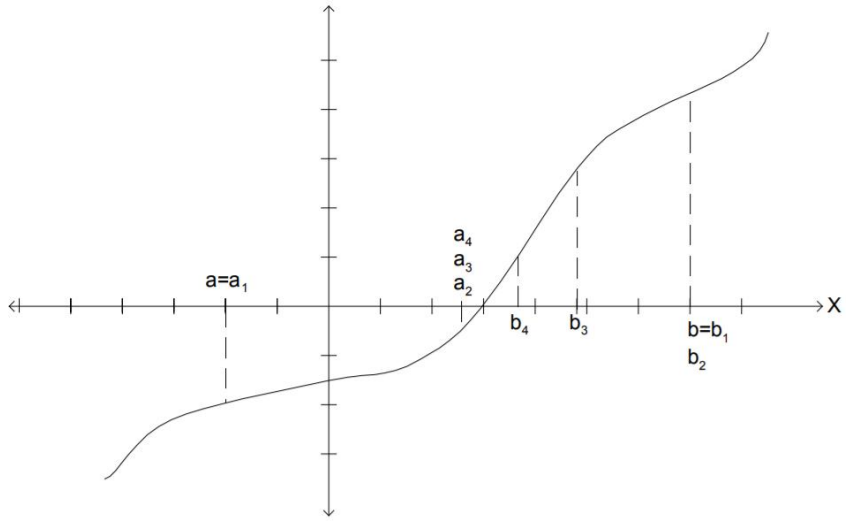
Man erkennt, dass der Graph die x-Achse zweimal schneidet. Damit liegen die Nullstellen der Funktion bei $x_1 = 0$ und $x_2 = 3{,}4567$.

Im Folgenden wird die Methode der Intervallhalbierung erklärt, da sie ohne höhere mathematische Kenntnisse (Differenzieren, …) auskommt.

Methode der Intervallhalbierung (Bisektion)

Das Verfahren der Intervallhalbierung ist zur Bestimmung der Nullstellen für stetige Funktionen (d.h. kein Knick oder Sprungstelle in der Funktion) geeignet.

Das Bestimmen einer Nullstelle einer Funktion $f(x)$ entspricht der Lösung der Gleichung $f(x) = 0$. Ist das Intervall $[a, b]$ bekannt, indem sich die Nullstelle befindet, so kann dieses halbiert werden. Aufgrund des Funktionswertes an der Halbierungsstelle kann entschieden werden, ob sich die Nullstelle in der oberen oder unteren Hälfte befindet. Dieses neue Intervall wird nach dem gleichen Verfahren wieder halbiert. Dieses auch als Bisektion bekannte Verfahren wird so oft wiederholt, bis das Intervall genügend klein ist:

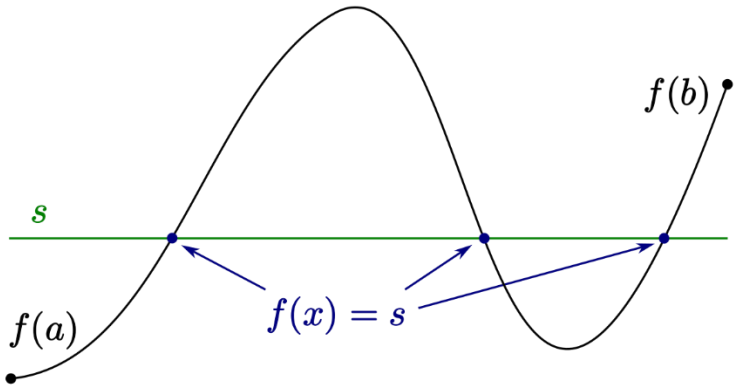


Die Lösung der Gleichung $f(x) = 0$ liegt nun in diesem Intervall und kann mit beliebiger Genauigkeit bestimmt werden. Als Startwert muss bei diesem Verfahren allein das Intervall $[a, b]$, indem die Nullstelle liegt, bekannt sein. Praktisch kann mit dieser Methode eine Genauigkeit von ca. 10^{-5} mit einfacher Genauigkeit (float) und ca. 10^{-10} mit doppelter Genauigkeit (double) auf einem Computer erreicht werden.

Mathematische Grundlagen

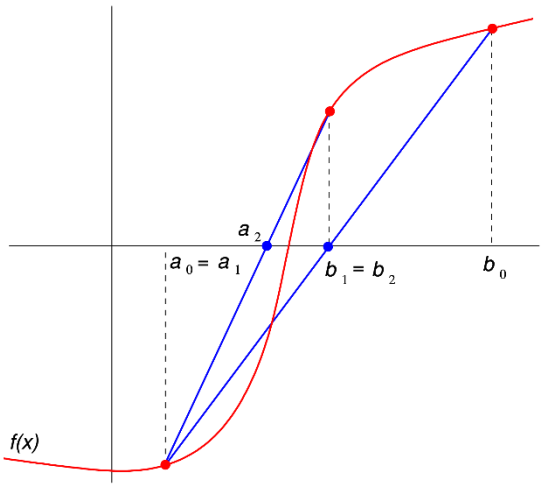
In der reellen Analysis ist der Zwischenwertsatz ein wichtiger Satz über den Wertebereich von stetigen Funktionen.

Der Zwischenwertsatz sagt aus, dass eine reelle Funktion f, die auf einem abgeschlossenen Intervall $[a, b]$ stetig ist, jeden Wert zwischen $f(a)$ und $f(b)$ annimmt. Haben insbesondere $f(a)$ und $f(b)$ verschiedene Vorzeichen, so garantiert der Zwischenwertsatz die Existenz von mindestens einer Nullstelle von f im offenen Intervall (a, b). Dieser Sonderfall ist als **Nullstellensatz von Bolzano** bekannt und nach Bernard Bolzano benannt.



Regula falsi

Neben der klassischen Intervallhalbierung gibt es auch noch die Möglichkeit, des ***Regula falsi***. Dabei wird eine Sekante durch die zwei Punkte $(a\ ,\ f(a))$ und $(b\ ,\ f(b))$ gelegt, und die Nullstelle der Sekante wird als neue Intervallgrenze betrachtet.



Der Algorithmus ist im Vergleich zur Intervallhalbierung exakt derselbe, das Bestimmen der neuen Grenze im Intervall (Schritt 3) wird aber durch folgende Formel bestimmt:

$$c = b - f(b) \cdot \frac{b - a}{f(b) - f(a)}$$

Ziel dieser Formel ist es anhand einer Sekantenfindung zwischen den beiden Punkten die Nullstelle schneller zu finden als bei der ursprünglichen Intervallhalbierung.

Newton-Raphson-Verfahren

Das Newton-Raphson-Verfahren ist eine Methode zur Bestimmung von Nullstellen einer Funktion $f(x)$. Es verwendet die Ableitung $f'(x)$, um die Nullstelle durch iterative Annäherung zu finden.

Vorgehensweise:

1. **Startwert wählen**: Wählen Sie einen Startwert x_0, der *möglichst* nahe an der tatsächlichen Nullstelle liegt.
2. **Iteration**: Berechnen Sie die nächste Näherung x_{n+1} mit der Formel: $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$
   Dabei ist x_n die aktuelle Näherung und x_{n+1} die nächste Näherung der Nullstelle.
3. **Wiederholen**: Wiederholen Sie den Iterationsschritt, bis die gewünschte Genauigkeit erreicht ist.

Beispiel:

Nehmen wir die Funktion $f(x) = x^2 - 2$ und deren Ableitung $f'(x) = 2x$. Wir suchen die Nullstelle dieser Funktion.

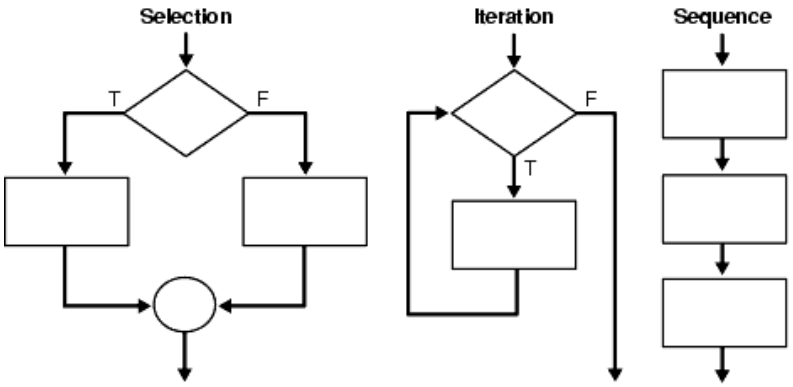
1. **Startwert**: Wählen wir $x_0 = 1$.
2. **Iteration**: Berechnen wir die nächste Näherung:
$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 1 - \frac{1^2 - 2}{2 \cdot 1} = 1 - \frac{-1}{2} = 1 + 0{,}5 = 1{,}5$$
3. **Wiederholen**: Führen wir den Iterationsschritt erneut durch:
$$x_2 = x_1 - x_0 - \frac{f(x_1)}{f'(x_1)} = 1{,}5 - \frac{1{,}5^2 - 2}{2 \cdot 1{,}5} = 1{,}5 - \frac{0{,}25}{3} = 1{,}4167$$

Durch wiederholte Anwendung dieser Schritte nähert sich immer weiter der tatsächlichen Nullstelle $\sqrt{2}$ an.

Programm Ablauf Plan

PAPs Entwurfshilfsmittel und dient dazu teilweise komplexe Aufgabenstellung in einer ersten Stufe für die Codierung vorzubereiten. PAPs sind gut dazu geeignet in Dokumentationen die wesentlichen Abläufe zu beschreiben. PAPs sind in der DIN 66001 geregelt.

In der strukturierten Programmierung bedient man sich folgender Grundlegenden Werkzeuge:



Die folgenden Code-Beispiele stammen aus Python.

Sequenz (engl. sequence, Aufeinanderfolgende Instruktionen): Aufeinander folgende, für die Aufgabe wichtige Operationen werden als Instruktionen ausformuliert. Dazu zählt z.B. das Zuweisen von Variablen.

```
1  a = 100
2  b = 50
3  c = a + b
4
5  print( c )
```

Alternative/Auswahl bzw. Wenn-Abfrage (engl. selection, Bedingte Ausführung aufgrund eines Entscheides): Je nach Resultat eines Entscheides wird eine Instruktion oder ein ganzer Block ausgeführt. Ist die Bedingung erfüllt (= TRUE) dann wird der Block ausgeführt, ist die Bedingung falsch (= FALSE) wird der Block in der ELSE-Anweisung ausgeführt.

```python
a = 100
b = 50

if a > b:
    print(f'{a} is greater than {b}')
else:
    print(f'{b} is greater than {a}')
```

Iteration (engl. iteration, Wiederholtes Ausführen von Instruktionen oder ganzen Programmteilen) Aufgrund einer Bedingung wird eine Instruktion oder ein Block mehrfach (n-mal) ausgeführt. Iterationen heißen auch *Schleifen*.

```python
for i in range(10):
    print( i )

i = 0
while True:
    print( i )
    i = i + 1
    if i > 9:
        break
```

Alle Beispiele liefern am Ende die Zahlen von 0 bis 9.

Eine Schleife kann jederzeit mit dem break Befehl unterbrochen werden.

In Python gibt es zusätzlich die for/else –Schleife. Läuft eine Schleife durch, ohne unterbrochen zu werden, wird der Befehl innerhalb des else ausgeführt.

```python
for i in range(10):
    print( i )
    if i > 12:
        print('break')
        break
else:
    print('no break')


for i in range(10):
    print( i )
    if i > 5:
        print('break')
        break
else:
    print('no break')
```

Aufgabenstellung

Aufgabe 1 (5 Punkte)

Sie möchten die Funktion $\sqrt[2]{n} = x$ für beliebige n iterativ lösen (z.B. $n = 25$ Bsp der Schulübung, wähle selbst eine Zahl). Schreibe Sie die Funktion so um, dass daraus ein Nullstellen-Problem $f(x) = 0$ wird. Finden Sie ein Intervall $x \in [a, b]$ sodass der Zwischenwertsatz $f(a) \cdot f(b) < 0 \rightarrow \exists\, x : f(x) = 0$ Anwendung findet. Erstelle Sie dann einen Graphen der Funktion (z.B. in Excel) und lies die Nullstelle ab. Ist die Funktion stetig?

Dies entspricht der Planungsphase Ihres Projekts. Sie visualisieren die Problemstellung und bekommen einen ersten Eindruck davon, wie Sie dieses umsetzen könnten.

Aufgabe 2 (5 Punkte)

Nutze Sie die umgeformte Funktion $f(x) = \cdots = 0$ aus Aufgabe 1 und führe die ersten 5 Iterationen der Bisektion für $n = [Katalognummer]$ **händisch** durch. Arbeite mit dem Intervall $[a, b] = [0, 2 \cdot Katalognummer]$

Dies entspricht der Planungsphase Ihres Projekts. Sie überlegen hier, wie der Algorithmus zur Lösung der Aufgabenstellung funktioniert.

Aufgabe 3 (5 Punkte)

Überlege Sie wie der Code aussehen könnte, der die Funktion aus Aufgabe 1 löst. Schreibe Sie dazu handschriftlich oder mit einer geeigneten Software einen Projektablaufplan (PAP nach DIN 66001) und verwende Sie die Elemente der strukturierten Programmierung (Sequenz, Auswahl und Iteration).

Dies entspricht der Planungsphase Ihres Projekts. Sie skizzieren den groben Ablauf Ihres Codes. Der PAP muss klar verständlich und lesbar sein. Die in der DIN 66001 enthaltenen Symbole müssen korrekt und sinnvoll benutzt werden. Punkte werden nur vergeben, wenn der PAP auch im Zuge der Programmierung umgesetzt wurde!

Aufgabe 4 (20 Punkte)

Wählen Sie ein **Vorgehensmodell**, das zu diesem Projekt passt. Argumentieren Sie warum Sich dafür entschieden haben.

Wählen Sie ein geeignetes Programmier-Paradigma (Strukturiert, Objektorientiert).

Betreiben Sie **Zeitmanagement** und schätzten Sie die benötigte Dauer (in Stunden ab) für die Umsetzung des Projekts ab.

Planen Sie ihr Projekt und erstellen Sie eine einfache **Work Breakdown Structure (WBS)** und erstellen sie ein einfaches **Gantt-Diagramm**. Überwachen Sie dabei den Soll- und Ist-Zustand!

Betreiben Sie **Ressourcenmanagement** und definieren Sie benötigte Ressourcen für Ihr Projekt (Personal, Hardware, Software, …).

Betrieben Sie **Risikomanagement**, d.h. überlegen Sie, welche Ereignisse den Erfolg Ihres Projekts (u.a. zeitgerechte Abgabe) beeinträchtigen könnte. Trennen Sie diese in innere und äußere Risiken.

Führen Sie eine **Veröffentlichungsprüfung** durch, d.h. testen Sie, ob Sie Ihre Software verkaufen dürfen (s. PSF-Lizenz, Urheberrecht und Marken).

Dies entspricht der Projektmanagementphase Ihres Projekts. Strukturieren Sie dieses Kapitel sinnvoll.

Aufgabe 5 (20 Punkte)

Programmieren (Python) Sie einen SOLVER für stetige Funktionen f mit einer Variablen x. Folgende Punkte müssen erfüllt werden:

- Nutze Sie dafür die Methode der Intervallhalbierung (**Bisektion**) als iteratives Lösungsverfahren für ein Nullstellen-Problem. Die Anfangswerte sind hier mit a und b gekennzeichnet und bilden das Intervall $[a, b]$. Mit der variablen c wird der Mittelwert des Intervalls $c = \frac{a+b}{2}$ gekennzeichnet. [25]
- Halten Sie den Code so dynamisch, sodass es möglichst leicht ist, eine beliebige andere Funktion lösen zu lassen. Z.B. könnte in der SOLVER selbst eine Funktion sein, die den Funktionswert mittels return zurückgibt. Es wird Ihnen für die weiteren Aufgaben helfen. Tipp: eval(). [5]
- **Teste** Sie den SOLVER mit der Wurzel-Funktion (s. Aufgabe 1) für $n = 25$, $n = 81$ und $n = 144$ und vergleichen Sie Ihre numerischen Näherung mit der analytischen Lösung! [5]

Strukturieren Sie ihren Code sinnvoll (Klassen, Funktionen, …), keine Monolithen und Redundanzen. **Kommentieren** Sie den Code ausreichend (so viel wie nötig). Führen Sie **Fehlerbehandlungen** ein (s. try … except). Halten Sie sich so gut wie möglich an die **PEP8** Programmiervorschrift (z.B. sinnvolle Benennung von Variablen).

<u>Abzüge</u>

- Keine Kommentare / docstrings [-5]
- Kein Typehinting [-5]
- Keine oder falsche Fehlerbehandlung [-5]

- Schwere strukturelle Fehler [-10]
- Nicht nachvollziehbare Abweichung vom PAP [-10]

Dies entspricht der Umsetzung Ihres Projekts und beinhaltet auch schon einen Softwaretest (= Softwarequalitätsmanagement). Setzen Sie Ihre Pläne und Skizzen um und erzeugen Sie Ihr Produkt!

Aufgabe 6 (20 Punkte)

Erstellen Sie eine alternative Lösungsmöglichkeit mit der Methode des *Regula falsi* **oder** dem *Newton-Raphson-Verfahren*. Erklärung zu den beiden Verfahren entnehmen Sie bitte den vorangegangenen Erklärungen. Gehe dabei wie zuvor bei Aufgabe 5 vor und **teste** die Ergebnisse mit den jeweiligen Zahlenwerten aus Aufgabe 1, 2, und 5.

Dies entspricht der Erweiterung Ihres Projekts durch zusätzliche Bedingungen und beinhaltet auch einen Softwaretest (= Softwarequalitätsmanagement). Erstellen Sie gegebenfalls zusätzliche Pläne und Skizzen und erweitern Sie Ihr Produkt!

Aufgabe 7 (10 Punkte)

Erstellen Sie eine grafische Aufbereitung der Nullstellenfindung mittles matplotlib und visulaisieren sie dort die Intervallhalbierung mit den verschiedenen Verfahren. Erstellen Sie folgende Diagramme.

- Die Diagramme sollten nach jedem Iterationsschritt ein Update erfahren und dabei den Bisektions-/Alternativprozess als eine Art Animation visualisieren. Tipp: Nutzen Sie Matplotlib Subplots sinnvoll, um zwei Axes in einer Figure darzustellen.
- Aktuelle Genauigkeit je Iterationsschritt. Der Graph sollte sich im Laufe der Iteration dem Wert 0 nähern, da wir die Nullstelle suchen.
- Aktuelle Lösung xx je Iterationsschritt. Zeigen Sie damit, wie sich der Wert xx langsam der Lösung annähert.

Dies entspricht der Umsetzunggs- und Testphase des Projekts in dem Sie die Ergebnisse noch graphisch aufbereiten und ihr Projekt noch weiterem Publik zugänglich machen wollen. Erweitern Sie es dazu um graphische Elemente!

Aufgabe 8 (5 Punkte)

Im Theorieteil wurde folgendes Polynom vorgestellt

$$P_4(x) = 2x + x^2 + 3x^3 - x^4$$

Finden Sie ein passendes Intervall $[a, b]$ um mit dem SOLVER die Nullstelle bei $x = 3{,}4567$ zu zeigen.

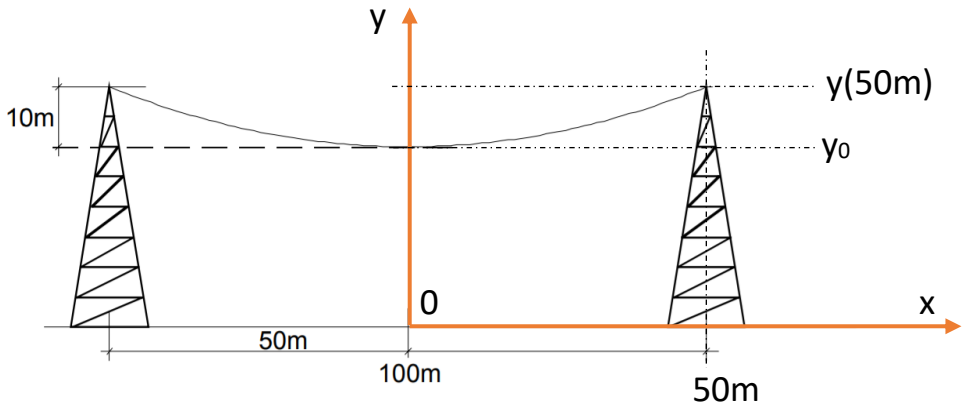
Führen Sie Tests bzgl. der Genauigkeit ε an dem Polynom durch.

- Wie viele Iterationsschritte sind für $\varepsilon = 10^{-2}$ notwendig?
- Wie viele Iterationsschritte sind für $\varepsilon = 10^{-8}$ notwendig?

Diese entspricht der Softwarequalitätsmanagement Phase. Prüfen Sie damit die Richtigkeit Ihrer Software. Überlegen Sie sich weiteren notwendigen Testfällen. Interpretieren Sie Ihr Ergebnis bei den Genauigkeitstests.

Aufgabe 9 (10 Punkte)

Eine elektrische Leitung wird zwischen zwei gleich hohen Masten, die $100\ \mathrm{m}$ auseinanderliegen, aufgespannt. Der maximale Durchhang der Leitung beträgt in der Mitte $10\ \mathrm{m}$.



Die Länge einer durchhängenden Kette oder eines Seils, kann durch die Formel

$$l = 2a \cdot \sinh\left(\frac{w}{2a}\right)$$

berechnet werden. Die Funktion $\sinh()$ ist eine der Hyperbelfunktionen (Hyperbelsinus). Sie sind die korrespondierenden trigonometrischen Funktionen an der Einheitshyperbel (also $x^2 - y^2 = 1$).[1] Dabei ist w der Abstand zwischen den Befestigungspunkten des Seils und a der Krümmungsradius des Seils am Scheitelpunkt. In diesem Fall ist $w = 100$. Der Krümmungsradius a ist unbekannt. Die Gleichung der Kettenlinie lautet[2]

$$y(x) = a \cdot \cosh\left(\frac{x - x_0}{a}\right) - a + y_0$$

Dabei sind x_0 und y_0 Verschiebungen des Scheitelpunkts in x- und y-Richtung (d.h. Abstände des Scheitelpunkts vom Ursprung des Koordinatensystems). Mit dieser Gleichung lässt sich a berechnen, doch ist das analytisch nicht möglich, da sie nicht nach a gelöst werden kann.

[1] https://de.wikipedia.org/wiki/Hyperbelfunktion
[2] http://www.physics.smu.edu/fattarus/catenary.html

Die Größe $x_0 = 0$, da die y-Koordinate durch den Scheitelpunkt verläuft. Die Größe y_0 (Abstand vom Boden $y = 0$ zum Scheitelpunkt des Seils) ist jedoch noch unbekannt. Eine Randbedingung ist uns aber bekannt, nämlich, dass für die Höhe der Strommasten

$$y(50\ \mathrm{m}) = y_0 + 10\ \mathrm{m}$$

gilt.

Wie groß ist die Länge l der Leitung? Wähle ein sinnvolles Intervall für die Anfangswerte. Benutze deinen SOLVER aus Aufgabe 5 & 6.

Tipp: Berechnen Sie den Krümmungsradius a durch iteratives Lösen und nutze ihn dann, um die Länge des Seils zu berechnen. Sie müssen vorher noch die Gleichung der Kettenlinie an einem bestimmten Punkt x (Welcher Punkte?) auswerten, der es dir ermöglicht, die Randbedingung einzusetzen.

Tipp: In Python finden Sie die Hyperbelfunktionen z.B. im Paket *numpy* oder *math*. z.B. für $X = \cosh(50)$

Dies entspricht der Anwendung Ihrer Software an einem realen Problem.

Abgabe

Ein ZIP-Verzeichnis mit:

- Die Lösungen aller Aufgaben sollen in **ein Protokoll** geschrieben werden.
    - WORD-Datei und PDF (mach aus deiner WORD-Datei zusätzlich ein PDF)
    - Deckblatt mit Programm-Titel, Name, Klasse, Schuljahr, Gegenstand, …
    - Alle Aufgaben sind **sauber** zu **dokumentieren**, d.h. schlüssige Rechenwege (inkl. Einheiten, wenn notwendig), kurze Beschreibung des Codes (gerne auch als Kommentar im Code selbst) mit Hilfe von Screenshots des Codes für alle Programmieraufgaben.
- Die **Quelldatei** des Codes inkl. kurzer Beschreibung wie das Programm ausgeführt wird. Die Abgabe ist so durchzuführen, dass jede Aufgabe einzeln getestet werden kann, ohne dass Eingriff in den Code notwendig ist.

Bitte teste deinen Code aus! **Lässt sich der Code nicht ausführen, gibt es keine Punkte für Code-Aufgaben. Der Code muss in der PowerShell / Terminal ausführbar sein.**

Jede Aufgabe MUSS einzeln testbar sein! Wir werden Ihren Code nicht so lange verändern, bis wir ihn testen können. Ist Ihr Code nicht möglichst leicht testbar, so gibt es keine Punkte für diesen Teil.

KEINE Gruppenarbeit! Du kannst dir Unterstützung holen, aber keinen Code oder Fragment davon kopieren oder abschreiben! Abgeschriebene Arbeiten werden zur Gänze mit 0 Punkten bewertet!

KEINE ganzen vorgefertigten Codes aus dem Internet! Code-Fragmente aus dem Internet (Stack Overflow, GitHub, …) sind mit entsprechendem Copyright des Autors zu versehen. Keine KI wie z.B. **ChatGPT**.

KEINE Punkte bei unpünktlicher Abgabe. Abgaben nur via E-Learning!

Viel Erfolg und Happy Hacking!