Computational Engineering und Robotik

Prof. Ph.D Jan Peters, J. Carvalho, P. Klink, P. Liu, S. Stark

Sommersemester 2020

7. Übung

Hinweise zu dieser Übung

- Für die Teilnahme an der Übung ist eine Anmeldung beim **Moodle** notwendig. Dort sind auf der Kursseite zusätzliche Informationen zur Veranstaltung und die Regelungen zur Anrechnung der Übungsleistungen in die Endnote aufgeführt:

  `https://moodle.tu-darmstadt.de/course/view.php?id=20214`

- Von dieser Übung werden die **1. Aufgabe** und die **Programmieraufgabe** bewertet. Lade zur Bearbeitung der Programmieraufgabe den aktuellen Code aus dem *git repository* herunter, welches im Moodle verlinkt ist.
- **Abgabe der schriftlichen Lösungen** zu den Übungsaufgaben bis **Montag, den 22.06.2019 um 12:00 Uhr** per Datei-Upload im Moodle.
- **Abgabe der Programmieraufgabe** bis **Montag, den 29.06.2019 um 12:00 Uhr** per Datei-Upload im Moodle.
- Lade im Moodle die Lösungen hoch als: **UebungXX.pdf** (z.B, für die Übung 04, Uebung04.pdf); **taskX.py** (z.B., für die zweite Programmieraufgabe task2.py). Gib deine Gruppennummer sowie die Namen und Matrikelnummern aller Beteiligten am Anfang der Datei an.
- Begründe alle deine Antworten. Falls du einen Scan deiner Antworten hochlädst, stelle sicher, dass sie lesbar sind.
- Stelle Fragen zu dieser Übung bitte im Moodle-Forum oder in den Sprechstunden.

1 Steife Differentialgleichungen (8 Punkte)

Betrachte für $\boldsymbol{x} \in \mathbb{R}^2$ und $t \geq 0$ die Anfangswertprobleme

$$\dot{\boldsymbol{x}}(t) = f_1(\boldsymbol{x}) = \begin{pmatrix} -4 & -2 \\ 1 & -2 \end{pmatrix} \boldsymbol{x}(t) + \begin{pmatrix} 4 \cdot \sin(t) \\ \sin(t) - 2 \cdot \cos(t) \end{pmatrix}, \quad \boldsymbol{x}(0) = \boldsymbol{x}_0 \tag{1}$$

und

$$\dot{\boldsymbol{x}}(t) = f_2(\boldsymbol{x}) = \begin{pmatrix} -400 & 0 \\ 1 & -2 \end{pmatrix} \boldsymbol{x}(t) + \begin{pmatrix} 400 \cdot \sin(t) \\ \sin(t) - 2 \cdot \cos(t) \end{pmatrix}, \quad \boldsymbol{x}(0) = \boldsymbol{x}_0. \tag{2}$$

mit dem Anfangswert $\boldsymbol{x}_0 = (0.1, 5)^T$.

a) Berechne die Steifigkeitskoeffizienten für beide Differentialgleichungen und interpretiere diese bezüglich der Steifheit der Differentialgleichungen.

b) Autonomisiere die Anfangswertprobleme (1) und (2), um die Abhängigkeit von der Zeit aufzulösen und gib die autonomisierten Anfangswertprobleme an.

c) Führe für beide autonomisierte Anfangswertprobleme drei Schritte des expliziten Eulerverfahrens mit Schrittweite $h = 0.1$ aus und gib die Ergebnisse an. Berechne die ersten beiden Schritte des autonomisierten Anfangswertproblems zu (1) handschriftlich. Für alle weiteren Berechnungsschritte kannst du *Python* oder ein anderes Numerikprogramm verwenden. Beurteile die Eignung des expliziten Eulerverfahrens für die Problemlösung.

d) Löse jetzt das autonomisierte Anfangswertproblem zu (2) erneut indem du das explizite Eulerverfahren mit einer kleineren Schrittweite von $h = 0.001$ ausführst. Gib die Ergebnisse für die Zeitpunkte 0.1 s, 0.2 s und 0.3 s an. Beurteile erneut die Eignung des expliziten Eulerverfahrens zur Problemlösung. Für die Berechnung kannst du *Python* oder ein anderes Numerikprogramm verwenden.

e) Ein Verfahrensschritt eines impliziten Lösungsverfahrens erfordert die Lösung eines nichtlinearen Nullstellenproblems. Stelle das Nullstellenproblem des autonomisierten Anfangswertproblems zu (2) für das implizite Eulerverfahren auf. Führe anschließend eine Iteration des Newton-Verfahrens zur Lösung des aufgestellten Nullstellenproblems mit Schrittweise $h = 0.1$ durch. Kannst du für dieses konkrete Problem das implizite Eulerverfahren auch analytisch, also ohne Verwendung des Newton-Verfahrens, lösen? Begründe deine Antwort.

2 Geometrische und numerische Lösung von Differentialgleichungen

Die Lösung des Anfangswertproblems

$$\dot{x}(t) = \sin\left(\frac{4}{3}t + 2\right) + \frac{1}{4}x(t) + \frac{2}{5}, \quad x(0) = x_0 = 0 \tag{3}$$

soll mit Hilfe der geforderten Integrationsverfahren geometrisch und numerisch bestimmt werden.

Verwende die Schrittweite $h = 2$ sowie für die geometrische Lösung das beigefügte Richtungsfeld der Differentialgleichung in (3). Mache in jedem Iterationsschritt die verwendeten Steigungen kenntlich. Runde die Zwischenergebnisse bei der numerischen Berechnung wenn notwendig auf mind. vier Nachkommastellen.

a) Konstruiere geometrisch die zwei Lösungspunkte x_1 und x_2 mit dem expliziten Euler-Verfahren. Überprüfe deine geometrische Lösung und gib den rechnerisch ermittelten Wert für x_1 an.

b) Konstruiere geometrisch die Lösungspunkte x_1 und x_2 mit dem impliziten Euler-Verfahren. Gib auch hier den rechnerisch ermittelten Wert x_1 nach der ersten Iteration an.

c) Konstruiere geometrisch die zwei Punkte x_1 und x_2 mit dem Heun-Verfahren. Gib auch hier den rechnerisch ermittelten Wert x_1 sowie die Zwischenwerte s_1 und s_2 an.

d) Konstruiere geometrisch die Lösungspunkte mit dem klassischen Runge-Kutta-Verfahren vierter Ordnung. Führe erneut geometrisch zwei Iterationsschritte aus und gib den rechnerisch ermittelten Wert x_1 sowie die Zwischenwerte s_1 bis s_4 an.

Programmieraufgabe P4 Simulationsstudie 3D-Drucker (5 = 3 + 2 Punkte)

Nachdem in der letzten Programmierübung die Regelung des 3D-Druckers implementiert wurde, soll in dieser Übung die numerische Lösung der Differentialgleichung, welche die Vorwärtsdynamik des Druckers beschreibt, genauer betrachtet werden. Bisher hat *CoppeliaSim* uns die Simulation des dynamischen Systems abgenommen. Ziel dieser Programmierübung ist es, verschiedene Integrationsverfahren in *Python* selbst zu implementieren und damit den zeitlichen Verlauf des Systemverhaltens bei gegebener Steuerung mit eigenen Mitteln zu simulieren.

1. Die Genauigkeit einer Simulation hängt unter anderem von dem verwendeten Verfahren und der Wahl der Schrittweite ab. Um die erreichbare Genauigkeit und den damit verbundenen Berechnungsaufwand abschätzen zu können, sollen vier gängige Verfahren untersucht werden: das **explizite Euler-Verfahren**, das **implizite Euler-Verfahren**, das **Heun-Verfahren** und das **klassische Runge-Kutta-Verfahren vierter Ordnung**.

- Implementiere die vier oben genannten numerischen Integrationsverfahren. Verwende dazu die Vorlage in `task4.py`.
- Alle Verfahren haben eine Methodensignatur der Form

  `x_k_1 = name_of_integrator(f, x_k, dt)`

  und bekommen als Eingabe eine autonome Differentialgleichung in Form des Funktionen-Handles **`f`**, welche das Vorwärtsdynamikmodell des zu simulierenden Systems beschreibt, den aktuellen Zustand des Systems in Form eines Spaltenvektors **`x_k`** sowie die gewünschte Schrittweite der Integration **`dt`** als Gleitkommazahl. Der Rückgabewert ist der Zustandsvektor **`x_k_1`** des Systems nach dem gegebenen Zeitschritt. Das Funktionen-Handle **`f`** hat die Methodensignatur **`dx_dk = f(x_k)`** und berechnet die zeitliche Änderung der Differentialgleichung an der gegebenen Position.
- Sofern ein numerisches Integrationsverfahren das Lösen nichtlinearer Gleichungssysteme erfordert, verwende dazu bitte das Newton-Verfahren, implementiert in Programmierübung 2c). Falls du keine eigene funktionsfähige Implementierung dieser Methode besitzt, kannst du die bereitgestellte Implementierung des Newton-Verfahrens in **`tasks/supplement.py`** verwenden. Die Jacobi-Matrix kann durch der Finite-Differenzen-Methode in Funktion **`fowardDQ`** aus **`tasks/supplement.py`** berechnet werden.
- Das klassische Runge-Kutta-Verfahren 4. Ordnung hat die Form:

$$
\begin{aligned}
\boldsymbol{s_1} &= f(\boldsymbol{x_k}) \\
\boldsymbol{s_2} &= f(\boldsymbol{x_k} + h/2 \cdot \boldsymbol{s_1}) \\
\boldsymbol{s_3} &= f(\boldsymbol{x_k} + h/2 \cdot \boldsymbol{s_2}) \\
\boldsymbol{s_4} &= f(\boldsymbol{x_k} + h \cdot \boldsymbol{s_3}) \\
\boldsymbol{x_{k+1}} &= \boldsymbol{x_k} + \frac{h}{6} \cdot (\boldsymbol{s_1} + 2\boldsymbol{s_2} + 2\boldsymbol{s_3} + \boldsymbol{s_4})
\end{aligned}
$$

2. Nun soll eine elliptische Bewegung des 3D-Druckers sowohl mit den zuvor implementierten Verfahren als auch mit *CoppeliaSim* simuliert und die jeweils resultierenden Abweichungen von der Solltrajektorie analysiert werden.

   Dazu wird statt einer Regelung eine einfache zyklische Steuerung der Form

$$\mathbf{F}(t) = \boldsymbol{a} \cdot \sin(t) + \boldsymbol{f_c} \ \in \mathbb{R}^3 \tag{4}$$

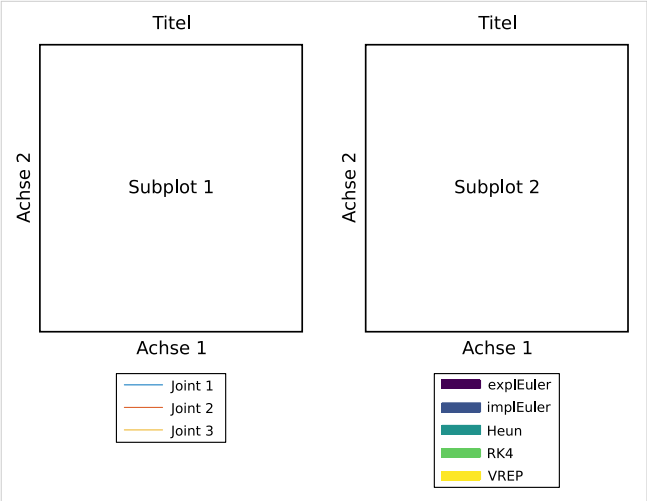
   angewendet, da sich das resultierende Vorwärtsdynamikmodell des Druckers dann analytisch lösen lässt. Diese analytische Lösung wird in der bereitgestellten **`class ControlJointsSolver`** berechnet und dient als Referenzlösung um den Fehler der verschiedenen Simulationsverfahren zu ermitteln. Die Ergebnisse sollen in zwei Plots übersichtlich dargestellt werden.

   Vervollständige dazu die Methode

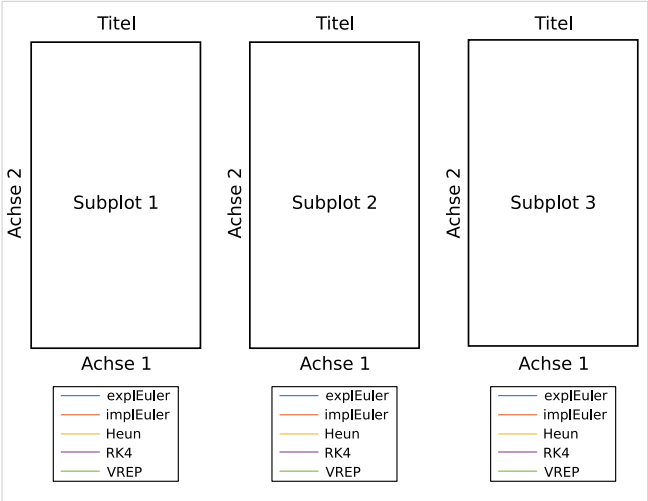
   `plot_solver_data(computation_times, actual_trajectories, solver, tmax=12, dt=0.01)`

   um zwei Plots zur Datenauswertung zu erstellen.

   - Eine Skizze mit dem Aufbau der beiden Plots ist mit Abbildung 1 gegeben. Es sollen folgende Informationen grafisch dargestellt werden:
     - **Plot 1:**
       - **Subplot 1:** Exakte Position der drei Gelenke in Gelenkkoordinaten (analytische Lösung), aufgetragen gegen die Zeit.
       - **Subplot 2:** Pro Schrittweite die für einen Simulationsdurchlauf benötigte Rechenzeit für jedes Verfahren.
     - **Plot 2:** Ein Subplot pro Schrittweite. Darin jeweils der Fehler (2-Norm über die 3 Gelenke) aller Verfahren, aufgetragen gegen die Zeit.
   - Der Datentyp von **`computation_times`** und **`actual_trajectories`** ist **`pandas.DataFrame`**. Weitere Informationen findest du auf dieser Website:
     https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html



(a) Plot 1



(b) Plot 2

Abbildung 1: Schematischer Aufbau der geforderten Plots.

- **`pandas`** bieten eine Plot-API, die die Standardkonvention für die Referenzierung der **`Matplotlib`** verwendet. Das Tutorial steht auf dieser Website:
  https://pandas.pydata.org/pandas-docs/stable/user_guide/visualization.html

- Installiere Pandas, falls du es bisher noch nicht installiert hast.

  ```
  conda install pandas
  ```

- Beachte, dass die der Methode übergebenen `2D-list` **`actual_trajectories`** beliebig lang sein dürfen.
- Achte auf eine aussagekräftige Beschriftung der Achsen und füge außerdem für *jeden* Subplot einen sinnvollen Titel und eine Legende hinzu. Die Farbe und die Reihenfolge kann von der Abbildung abweichen. Positioniere die Legenden wie in Abbildung 1 dargestellt.
- Stelle dabei sicher, dass Werte ≤ 0 nicht geplottet werden und wähle zur besseren Vergleichbarkeit für alle drei Subplots dieselben y-Achsenlimits.

Hinweise zur Programmieraufgabe

- Für jede Programmierenaufgabe, lade das git *repository*, das im Moodle verlinkt ist, und ziehe die neuesten Updates mit: `git pull`.
- Auf der Startseite der Veranstaltung im Moodle ist eine kurze Einführung in Python und CoppeliaSim verlinkt, welche die wichtigsten Grundlagen zur Verwendung der Programme erläutert.
- Folge der Anleitung zur Installation und Ausführung von Programmen im git *repository*.
- Für die Aufgabe brauchst du nur die Dateien im Ordner *tasks* zu verändern.
- Zur Überprüfung deiner Lösung der einzelnen Teilaufgaben findest du im Ordner `tests` öffentliche Testfälle. Diese können durch den Aufruf des Befehls `make test4` (für task4) im Hauptordner ausgeführt werden. Zur Bewertung werden neben diesen Tests **weitere** Testfälle überprüft!
- Neben den mitgelieferten Testfällen kannst du die Simulation mit `make p4` (für task4) starten. Weitere Informationen findest du im git *repository*.
- Für die Abgabe der Programmieraufgabe lade **nur die zu bearbeitenden Dateien** im Moodle hoch, die sich im Ordner `tasks` befinden.

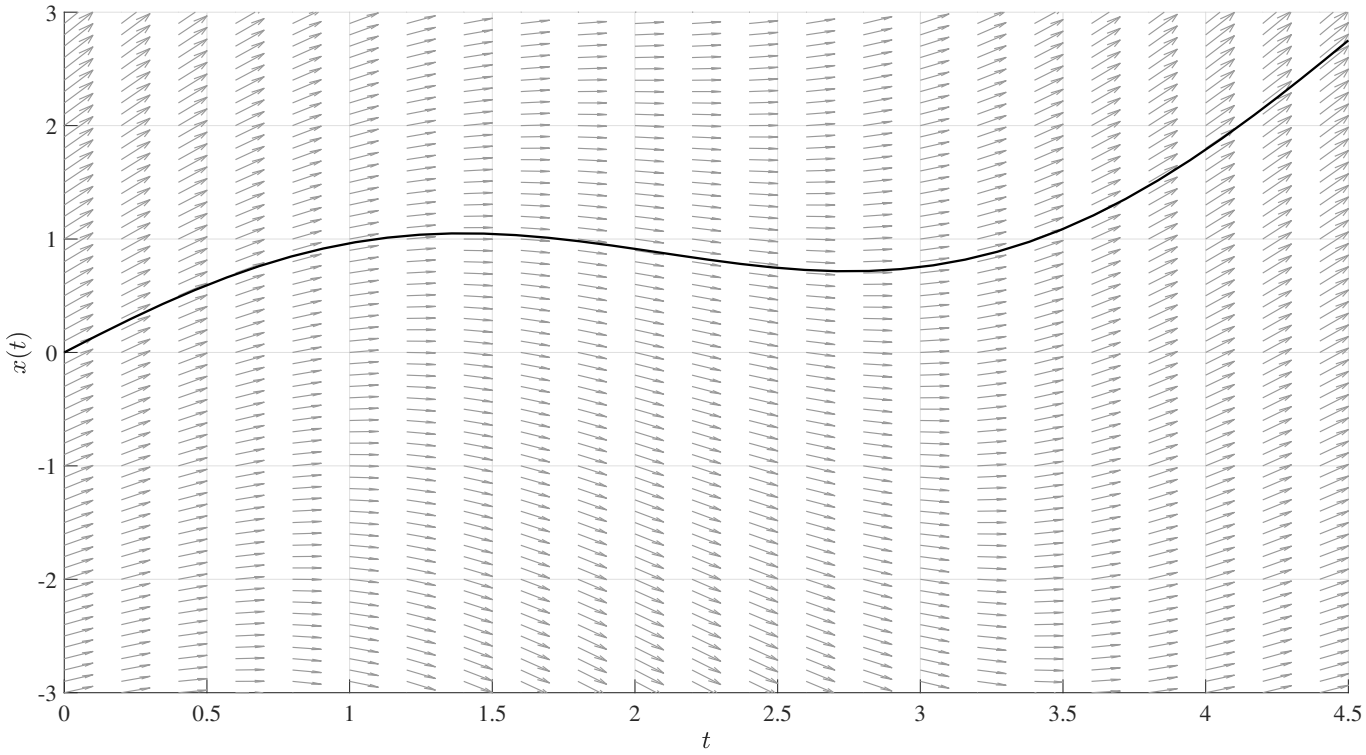
- Gib als Kommentar am Anfang jeder Datei im Ordner `tasks` die Namen und Matrikelnummern aller deiner Gruppenmitglieder an.
- **Es ist ausreichend, wenn EIN Gruppenmitglied die Lösung einsendet. Alle angegebenen Gruppenmitglieder einer Abgabe bekommen dann dieselbe Bewertung.**

Hinweis zu wissenschaftlichem Arbeiten

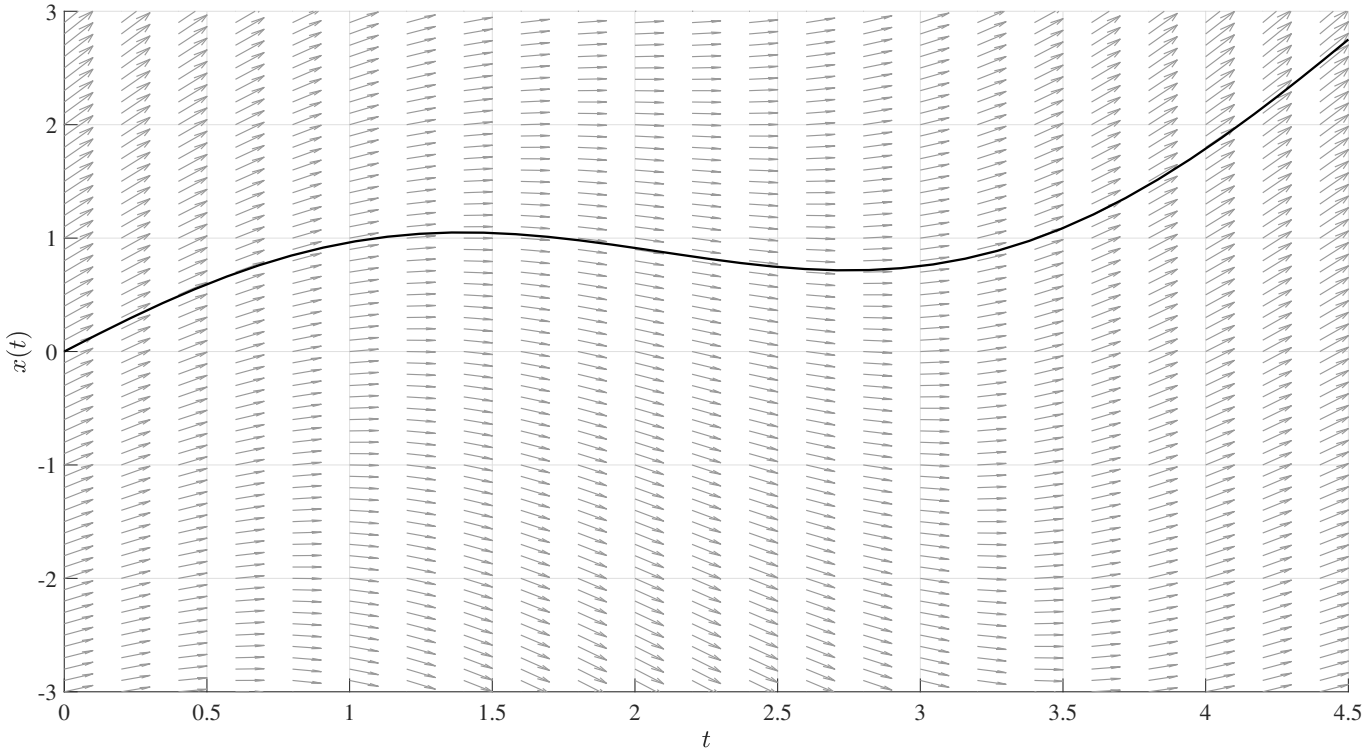
Es ist nicht gestattet, Lösungen anderer Personen als die der Gruppenmitglieder als Lösung der Aufgabe abzugeben. Des Weiteren müssen alle zur Lösungsfindung verwendeten, darüber hinausgehenden, relevanten Quellen explizit angegeben werden. Dem widersprechendes Handeln ist Plagiarismus und ist ein ernster Verstoß gegen die Grundlagen des wissenschaftlichen Arbeitens, das ernsthafte Konsequenzen bis hin zur Exmatrikulation haben kann.

Vorlage: Richtungsfelder zu Aufgabe 2

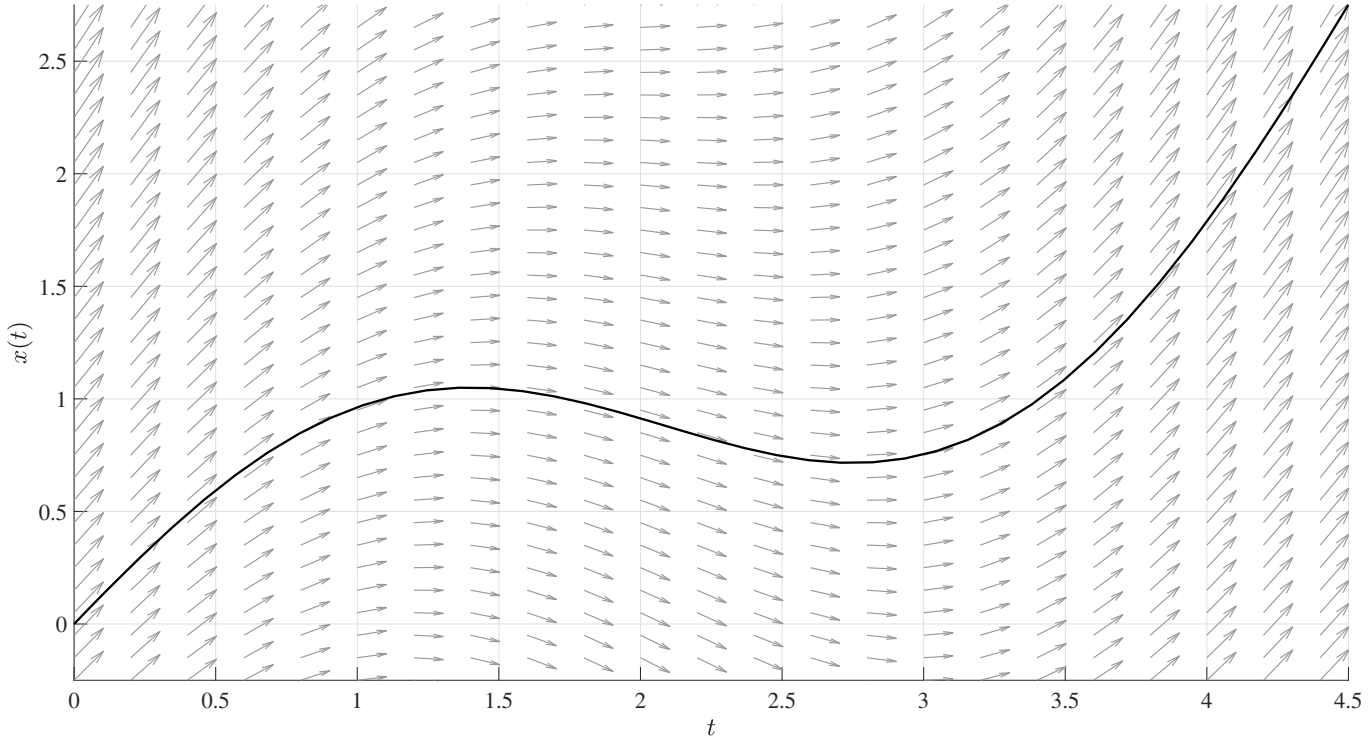
Vorlage geometrische Konstuktion explizites Euler-Verfahren



Vorlage geometrische Konstuktion implizites Euler-Verfahren



Vorlage geometrische Konstuktion Heun-Verfahren



Vorlage geometrische Konstuktion Runge-Kutta-Verfahren