Einführung in Computational Engineering

http://lehre.ias.informatik.tu-darmstadt.de/ComputationalEngineering/ComputationalEngineering

Einführung in Computational Engineering – Vorlesung 1

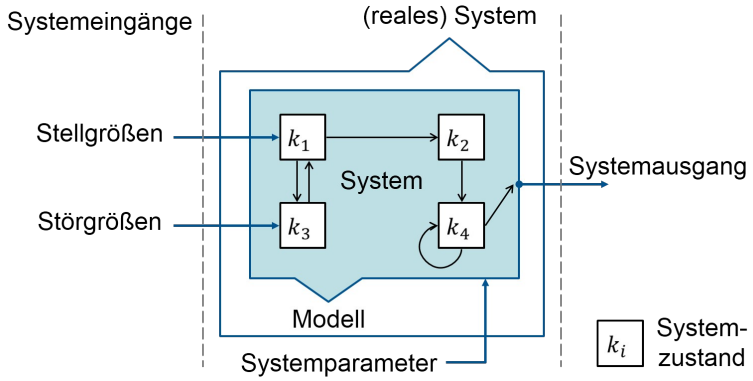
In dieser Vorlesung beschäftigen wir uns damit, was ein Modell und was Simulation ist, sowie diese aufgebaut sind und wie man mit diesen arbeitet. Außerdem werden wichtige Begrifflichkeiten für das Feld der Simulation und für Modelle eingeführt werden. Zudem werden auch noch Eigenschaften von Modellen eingeführt und erklärt, welche für den weiteren Verlauf der Vorlesung relevant für das Verständnis sind.

Simulation und Modell – wofür?

Zuerst wollen wir uns mit der Frage beschäftigen, wofür Modelle oder Simulationen überhaupt notwendig sind. Simulation von Modellen oder Modelle sind genau dann sinnvoll, wenn ein konkretes Problem vorliegt, welches nur mit hohem (z.B. finanziellen) Aufwand oder gar nicht umgesetzt werden kann. In der Vorlesung wird konkret ein Warteschlangenmodell behandelt, welches in den nächsten beiden Vorlesungen zudem vertieft wird.

Intuition zu Modellen und Simulation

Unter einem Modell kann man sich ein abstraktes Konstrukt vorstellen, welches einen Ausschnitt oder Teilausschnitt der realen Welt betrachtet. Bei der Modellbildung werden für die Simulation unwichtige Details außer acht gelassen, da diese den Prozess unnötig komplex machen würden.



Die Simulation ist der Prozess bei dem man das erstellte abstrakte Modell gegeben dem spezifischem Problem auf die Probe stellt um seine Simulationsziele zu erreichen. Die Simulationsziele sind im Allgemeinem (wie auch in den Vorlesung vorgestellt):

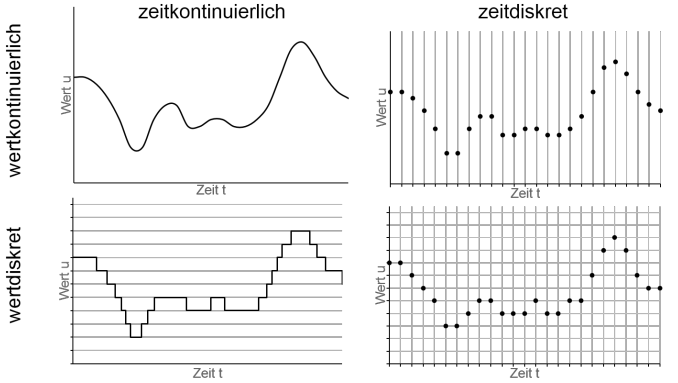
•ein bekanntes Szenario zu verstehen bzw. nachzuvollziehen,

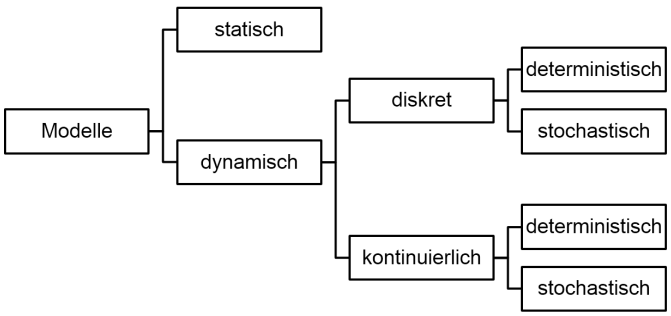
•ein bekanntes Szenario zu optimieren oder

•ein unbekanntes Szenario vorherzusagen.

Modellarten

Modelle werden häufig als diskret oder kontinuierlich aufgefasst. So kann ein Modell zum Beispiel zu einem gewissen Zeitpunkt einen bestimmten Wert haben. Demnach lassen sich Zeit und Wert nach diesem Raster klassifizieren.

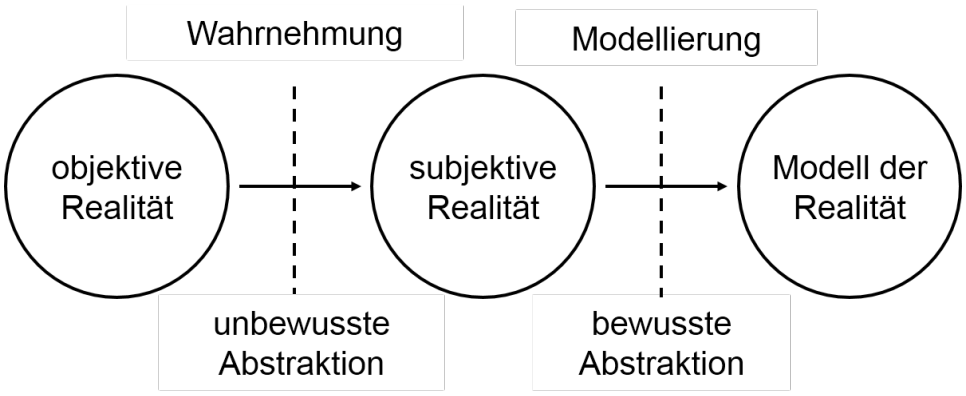




Die Simulation kann dabei unterschiedlich aufgebaut sein. Die Zustände selbst können diskret oder kontinuierlich sein und die Übergänge deterministischer bzw. stochastischer Natur sein. Des Weiteren können einzelne Schritte einer Simulation unterschiedlich verteilt sein, d.h. dass zum Beispiel diese Zustände kontinuierlich, diskret äquidistant, disrekt nicht äquidistant, usw. verteilt sind. Die Verteilung wird hier [1] noch einmal genauer erklärt.

Wie arbeitet man mit Modellen?

Zum Arbeiten mit Modellen ist es vorerst einmal wichtig zu wissen, wie man Modelle herleiten kann. Dazu wollen wir mit der folgenden Grafik einen möglichen Entstehungsprozess eines Modells aufzeigen. Hierbei werden formale Modelle mithilfe von mathematischer Modellierung aufgestellt. Die Anforderungen an ein solches Modell sind wie oben auch beschrieben der verringerte Detailgrad, sprich ein idealisiertes Modell.



Man betrachtet hier (beim Modellieren) dann noch zwei Größen:
•Qualitativ – Identifizierung der relevanten Kenngrößen

•Quantitativ – konkrete Größe der Abhängigkeiten

Bewertung von Modellen

Zur Bewertung von Modellen wird oft Validierung zu Rate gezogen. Verschiedene Validierungsformen sind:
•Vergleich unterschiedlicher Modelle bei gleicher Simulation

•Plausibilitätstest – Vergleich der Simulation mit bisher bekannten Ergebnissen und Theorien (z.B. aus der Vergangenheit)

•A-posteriori Betrachtungen – Bewertung aufgrund von Beobachtung von Systemen aus der Realität

•Modellvergleich – Vergleich verschiedener Modelle (so z.B. für die Optimierung)

Unter anderem ist bei der Bewertung gegebenenfalls auch auf Sicherheit, sowie auf Genauigkeit von Modellen zu achten. Besonders bei Sicherheitskritischen Systemen wie in der Automobilindustrie ist die Simulation notwendig.

Einführung in Computational Engineering – Vorlesung 2

Autoren: Andrej Felde und Thomas Hesse

In dieser Vorlesung wiederholen wir die Begrifflichkeit diskreter Modelle und beschäftigen uns anschließend hauptsächlich mit der ereignisdiskreten Simulation eines Warteschlangenmodells. Einen Überblick über alle relevanten Begriffe zur ereignisdiskreten Simulation findet man hier [1]. Hier wird zudem ein Algorithmus zur ereignisdiskreten Simulation eines Warteschlangenmodells vorgestellt. In Vorlesung 3 [2] wollen wir dann das Warteschlangenmodell verallgemeinern. Danach schauen wir uns ein weiteres Modell, das sogenannte Petrinetz, zur (ereignis)diskreten Modellierung an. Anschließend definieren wir relevante Eigenschaften für Petrinetze.

Ereignisdiskrete Simulation eines Warteschlangenmodells

Nachdem in der Vorlesung diskrete Modelle wiederholt wurden, wird hier die Kenntnis darüber vorausgesetzt. (Bei Problemen bitte erneut mitVorlesung 1 [3] auseinandersetzen.) Wir nehmen Bezug auf die vorherige Vorlesung 1 [3] in der wir darüber gesprochen haben, wie wir ein Modell für eine konkrete Problemstellung herleiten. Im folgendem betrachten wir das konkrete Beispiel eines Supermarktes.

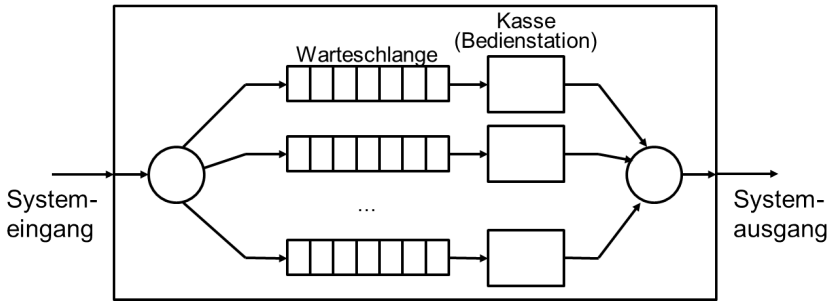


Abbildung 1. Abstrahierte Darstellung von Warteschlangen mit Kassen. Der Sachverhalt in diesem Bild modelliert das System an sich.

Zuerst abstrahieren wir den Sachverhalt und blenden damit unnötige Details aus. Weiter versuchen wir für die Simulation wichtige Objektklassen und Attribute zu identifizieren und zu benennen (mathematische Modellbildung). Wichtige Objektklassen wären für unser Modell *Kunde/in, Kasse* und *Kassierer/in*. Möglicherweise relevante Attribute

für Kunden wären dann z.B. Anzahl der Artikel, jeweils Eintrittszeitpunkt und Austrittszeitpunkt in und aus dem System. Etwas einfacher wären die Attribute für die Kasse (offen oder geschlossen) und für die Kassierer/innen (routiniert oder langsam). In Abbildung 1. erkennen wir, dass die jeweilige Kasse ihre Warteschlange sequentiell abarbeitet. Somit könnten wir für jede einzelne Kasse z.B. den Durchsatz erhöhen und somit das System optimieren, indem man mittels Simulation herausfindet wie viele Kassen bei wie vielen Kunden gleichzeitig geöffnet sein müssten.

Discrete-event simulation (DES Algorithmus)

Der Algorithmus zur ereignisdiskreten Simulation bietet sich sehr gut für die Simulation von Warteschlangenmodellen an. In der Simulation selbst geht man davon aus, dass zwischen den Ereignissen keine Änderungen geschehen. Daher kann man von einem ereignisdiskreten Zeitpunkt zum nächsten springen. Dies wird in der folgenden Abbildung 2. dargestellt. Den Algorithmus selbst, finden Sie hier [4].

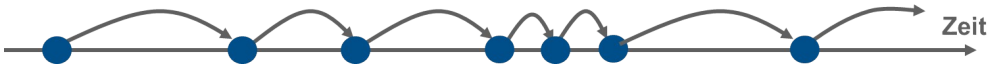
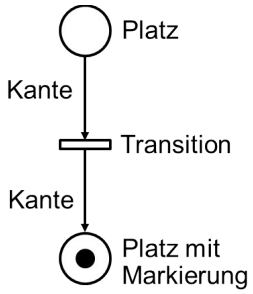


Abbildung 2. Ereignisdiskrete "'Zustandssprünge'" auf der Zeitachse. Die "'Zustandssprünge'" sind hier als Zustandsübergänge zu verstehen.

Petrinetze

Petrinetze können diskreter oder kontinuierlicher Natur sein. In dieser Vorlesung betrachten wir jedoch nur den diskreten Fall (insbesondere den ereignisdiskreten Fall) von Petrinetzen um einen guten Einsteig in das Thema zu gewährleisten.
Ein Petrinetz ist ein System bestehend aus dem folgendem Tupel $PN = (P, T, A, K, M_0)$. Die Bedeutung der einzelnen Parameter wird in folgender



Auflistung erklärt. Dafür sei $n, m, l \in \mathbb{N}$.
•P – endliche Menge aller Plätze $p_1, p_2, \dots, p_n$

•T – endliche Menge aller Transitionen $t_1, t_2, \dots, t_m$

•A – endliche Menge gerichteter Kanten $a_1, a_2, \dots, a_l$

•K – Menge der Platzkapazitäten $k_1, k_2, \dots, k_n$

•M_0 – Anfangsmarkierungen der Plätze $m'_1, m'_2, \dots, m'_n$

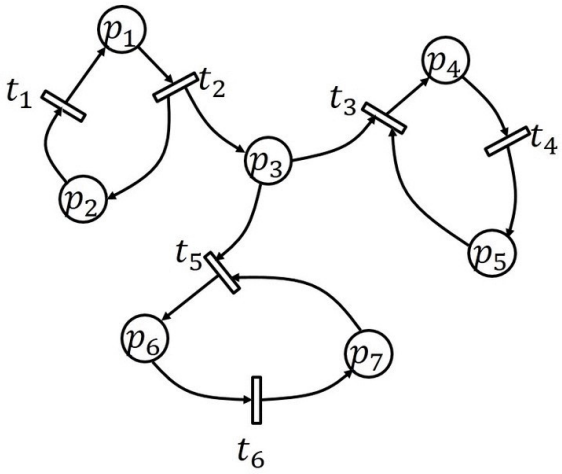
Der Zustand eines Petrinetzes wird durch die Menge der Markierungen M ausgedrückt. Zudem hat ein Petrinetz verschiedene Eigenschaften, welche hier [5] genauer erklärt werden. Nun wollen wir uns Petrinetze etwas praktischer anschauen, um zu verstehen wie diese funktionieren. Es ist wichtig zu wissen, dass bei einem Petrinetz eine Transition nur dann schalten kann (auch "feuern" genannt), wenn alle Eingangsplätze (damit sind Plätze gemeint, die eine Kante zur Transition besitzen) mit mindestens einer Markierung belegt sind. Anschließend wird aus **jedem** Eingangsplatz eine Markierung entnommen und in**jedem** Ausgangsplatz eine Markierung hinzugefügt.
•Eingangsplätze („fan-in") – Plätze, die eine Kante zu der Transition besitzen

•Ausgangsplätze („fan-out") – Plätze, mit einer Kante von der Transition zum Platz

Hinweis: Eingangs- bzw. Ausgangsplätze beziehen sich stets auf eine bestimmte Transition. Jede Transition kann verschiedene Eingangs- und Ausgangsplätze haben.

Petrinetz Beispiel

Als Beispiel betrachte man folgendes Petrinetz:



Man sieht ein Petrinetz mit sieben Plätzen und sechs Transitionen. Zur Initialisierung sind die Plätze p_1, p_5 und p_7 markiert. Bei Initialisierung gilt also $m_0 = [1000101]^T$. Feuert nun t_2 muss eine Markierung von p_1 entfernt und jeweils eine Markierung bei p_2 und p_3 hinzugefügt werden. Nun können t_1, t_3 als auch t_5 feuern. Im Beispiel feuert t_5. Dies führt dazu, dass eine Markierung bei den Plätzen p_3 und p_7 entfernt und bei p_6 hinzugefügt wird. Für t_5 sind p_3 und p_7 Eingangsplätze und p_6 ein Ausgangsplatz.

Inzidenzenmatrix

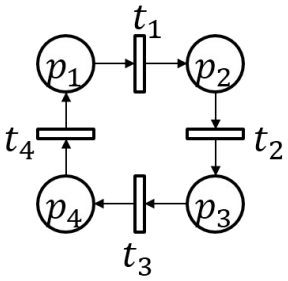
Eine Inzidentenmatrix W beschreibt die Beziehung der Knoten und Kanten in einem Petrinetz. Genauer gesagt beschreibt dieses welche Plätze p_i nach dem Schalten einer Transition t_i eine weitere Markierung erhalten und von welchen Plätzen eine Markierung entnommen wird. Die Spalten einer Inzidenzenmatrix stehen für die jeweiligen Transitionen und die Zeilen für die jeweiligen Plätze eines Petrinetzes. Grundsätzlich kann man eine Inzidenzenmatrix in folgende Matrizen aufteilen:

•Matrix W^+ („post-weights“) besteht aus Verbindungen $t_i \rightarrow p_j$

•Matrix W^- („pre-weights“) besteht aus Verbindungen $p_i \rightarrow t_j$

Intuitiv beschreibt die Matrix W^+ welche Plätze nach dem Schalten einer Transition eine weitere Markierung erhalten und Matrix W^- welche Plätze vor dem Schalten einer Matrix mindestens eine Markierung enthalten haben. Aus der Differenz der „post-weights“ und der „pre-weights“ ergibt sich die Inzidenzenmatrix $W = W^+ - W^-$.

Beispiel



Allgemein lässt sich die Größe der Inzidenzenmatrix über die Anzahl der Transitionen und Plätze ermitteln. Die Einträge ergeben sich über den sogenannten „fan-in" und „fan-out". Das Petrinetz besitzt jeweils vier Plätze und vier Transitionen. Damit ergibt sich eine 4x4-Matrix mit folgenden Werten:

$$W^+ = \begin{matrix} \\ p_1 \\ p_2 \\ p_3 \\ p_4 \end{matrix} \begin{matrix} \begin{matrix} t_1 & t_2 & t_3 & t_4 \end{matrix} \\ \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \end{matrix} \qquad W^- = \begin{matrix} \\ p_1 \\ p_2 \\ p_3 \\ p_4 \end{matrix} \begin{matrix} \begin{matrix} t_1 & t_2 & t_3 & t_4 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{matrix}$$

Und wie wir gesehen haben, ergibt sich dann aus der Differenz der „post-weights" und der „pre-weights" dann die resultierende Inzidenzenmatrix:

$$W = W^+ - W^- = \begin{bmatrix} -1 & 0 & 0 & 1 \\ 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix}$$

Einführung in Computational Engineering – Vorlesung 3

Autoren: Andrej Felde und Thomas Hesse

Zufallszahlen und lineare Kongruenz

Bei dem linearen Kongruenzgenerator handelt es sich um eine Methode um Zufallszahlen im Rechner zu erzeugen. Zufallszahlen sind für Simulationen insofern relevant, da man meist viele Eingaben braucht und diese aufgrund von Empirie durchzuführen ist oft zu zeit-intensiv oder zu kosten-intensiv. Daher betrachten wir zufällig generierte Werte, welche dann anschließend (und auf der Problemstellung basierend) durch eine spezifische Wahrscheinlichkeitsverteilung angepasst werden kann. Daher zeigen wir jetzt erst einmal wie man über den linearen Kongruenzgenerator Zufallszahlen erzeugt:

1.Wahl eines Startwertes, dem sogenannten *seed*. Mathematisch notieren wir dies als $x_0 = \text{seed}$.

2.Erzeugen von mehreren Zufallswerten basierend auf dem vorherigem Zufallswert mit $x_{k+1} = (a \cdot x_k + c) \mod m$.

Dabei ist a und c einmal fest gewählt und jeweilig ein Multiplikator und ein Inkrement, welche zusätzlich noch mehr Zufall erzeugen sollen. $m$beschreibt hier den Modulus. Der interessierte Leser entdeckt dabei auch natürlich, dass es sich hierbei um $m-1$ viele Äquivalenzklassen handelt und somit nicht mehr Zufallszahlen als Äquivalenzklassen generiert werden können. Anschließend können diese Werte nun einer Verteilung zugeordnet werden. Außerdem lässt sich mithilfe des linearen Kongruenzgenerator eine Zufallsvariable X definieren.

Wahrscheinlichkeit

Die Wahrscheinlichkeit für das Eintreten eines Ereignisses A lässt sich durch die Wahrscheinlichkeitsfunktion $Pr[A = X]$ mit Zufallsvariable Xbzw. $Pr[A]$ beschreiben. Für die Wahrscheinlichkeitsfunktion gilt $Pr[A] \in [0,1]$ und die Summe aller Wahrscheinlichkeiten $Pr[A]$ ergibt immer 1.

- $Pr[A]$ – Wahrscheinlichkeit des Ereignisses A.
- $Pr[A \wedge B]$ – Wahrscheinlichkeit, dass beide Ereignisse A und B eintreten.
- $Pr[A|B]$ – Wahrscheinlichkeit, dass A eintritt, unter der Bedingung, dass B bereits eingetreten ist. *(bedingte Wahrscheinlichkeit)*
- $Pr[A|B] \cdot Pr[B] = Pr[A \wedge B]$ – Multiplikationssatz für bedingte Wahrscheinlichkeiten.

Sind zwei Ereignisse A und B voneinander unabhängig und gilt für die einzelnen Wahrscheinlichkeiten $Pr[A], Pr[B] > 0$ dann gilt:

- $Pr[A|B] = Pr[A]$

Verteilungen

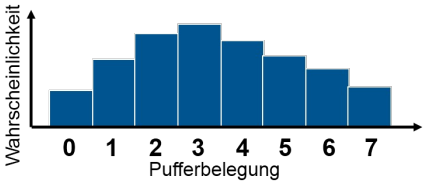
Die Verteilung einer Zufallsvariable X kann mit einer Verteilungsfunktion $F(x)$ beschrieben werden. Alle möglichen Wahrscheinlichkeiten einer Zufallsvariable können dann mithilfe der Verteilungsfunktion berechnet werden. Verteilungsfunktionen können sowohl diskret als auch kontinuierlich verteilte Zufallsvariablen beschreiben und es gilt allgemein:

- $F(x) = Pr[X \leq x]$
- $F(b) - F(a) = Pr[a < X \leq b]$
- $F(-\infty) = 0$

- $F(\infty) = 1$

Für diskret verteilte Zufallsvariablen kann man die einzelnen Wahrscheinlichkeiten $p_k = Pr[x_k = X]$ angeben. Für die Verteilungsfunktion folgt:

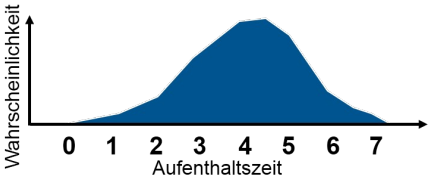
- $F(x_k) = \sum_{i=0}^{k} p_i$



Ein Beispiel für eine diskret verteilte Zufallsvariable wäre die Pufferbelegung.

Für kontinuierlich verteilte Zufallsvariablen kann man die Verteilungsfunktion über die Wahrscheinlichkeitsdichte $f(x)dx = Pr[x < X \leq x + dx]$ angeben. Für die Verteilungsfunktion folgt dann:

- $F(x) = \int_{-\infty}^{x} f(s)ds$



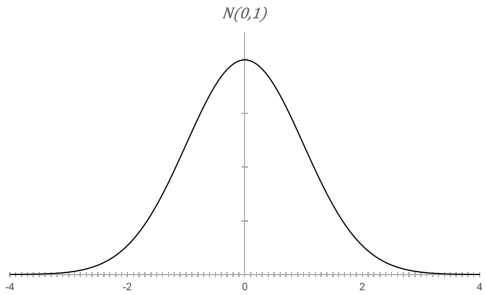
Ein Beispiel für eine kontinuierlich verteilte Zufallsvariable wäre die Aufenthaltszeit einer Entität im Puffer.

Normalverteilung

Abweichungen vom Mittelwert lassen sich besonders gut durch die Normalverteilung beschreiben. Eine Zufallsvariable X ist normalverteilt mit Parametern μ und σ^2, kurz $N(\mu, \sigma^2)$. Ist eine Zufallsvariable X normalverteilt, so schreibt man auch $X \sim N(\mu, \sigma^2)$. Die Dichtefunktion der Normalverteilung lautet:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}, \qquad x \in \mathbb{R}$$

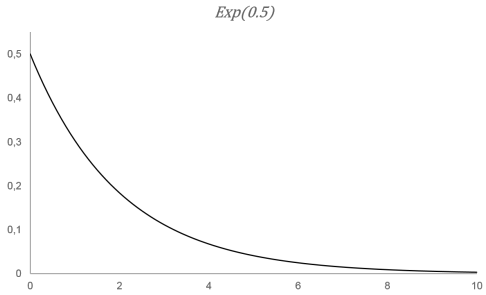
Dabei steht μ für den Erwartungswert und σ^2 für die Varianz. Sind der Erwartungswert $\mu = 0$ und die Varianz $\sigma^2 = 1$ spricht man von einer Standard-Normalverteilung.



Exponentialverteilung

Nur die exponentielle Verteilungsfunktion ist ohne Gedächtnis mit Markov Eigenschaft. Daher eignet sie sich besonders zufällige Zeitintervalle zu beschreiben. Eine Zufallsvariable X mit Parameter $\lambda > 0$, kurz $Exp(\lambda)$ heißt exponentialverteilt. Ist eine Zufallsvariable X exponentialverteilt, so schreibt man auch $X \sim Exp(\lambda)$. Die Dichtefunktion der Exponentialverteilung lautet:

$$f(x) = \begin{cases} 0 & x < 0 \\ \lambda \cdot e^{-\lambda x} & x \geq 0 \end{cases}$$



Allgemeines Warteschlangenmodell

Das Warteschlangenmodell (Warteschlangentheorie) kann in verschiedensten Gebieten genutzt werden. Warteschlangen treten überall im Alltag auf. Beispiele dafür wären:

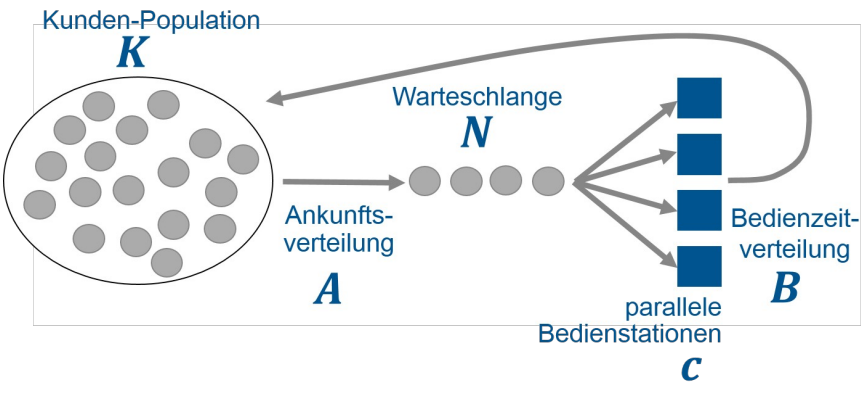
•Studenten in der Mensa

•Autos im Stau oder vor der Ampel

•Kunden vor der Supermarktkasse

•...

Die genannten Beispiele haben alle bestimmte Gemeinsamkeiten. Ein Student, Auto, Kunde oder allgemein eine Entität möchte von einer Bedienstation bearbeitet werden. In unseren



Beispielen wären das die Essensausgabe in der Mensa, eine Ampel oder die Kasse im Supermarkt. Ist die Bedienstation belegt muss die Entität in einer Warteschlange oder im Stau auf die Bearbeitung warten. Ziel des Warteschlangenmodells ist es den Ablauf zu optimieren oder nachzuvollziehen, bei bekanntem Szenario oder bei unbekanntem Szenario eben solch einen Sachverhalt vorherzusagen. Bei einem Supermarkt könnte man so feststellen was die optimale Anzahl an Kassen wäre, ohne zu viel für Kassierer/innen zu bezahlen bzw. ohne die Kunden zu lange warten zu lassen. Oder man kann vorerst feststellen, wie dieses Szenario im Allgemeinem abläuft. Interessant ist auch solch ein Szenario vorherzusagen, da man im richtigem Leben eventuell nicht so viel Geld hat zuerst einen Testlauf durchzuführen. Allgemein kann man das Warteschlangenmodell in folgende Elemente aufteilen:

- Kundenpopulation ***K*** – besagt wie viele Entitäten im System vorhanden sind. Bei den meisten Beispielen in der Vorlesung ist die Kundenpopulation unbeschränkt.
- Ankunftsverteilung ***A*** – beschreibt wie schnell bzw. in welcher Art die Entitäten ankommen. Die Ankunftsverteilung kann durch verschiedene bereits bekannte Verteilungen beschrieben werden. Ein Beispiel wäre die Exponentialverteilung.
- Warteschlange ***N*** – gibt die Länge der Warteschlange wieder. Die Größe der Warteschlange kann beschränkt werden. Kommt eine Entität bei vollen Warteschlange an geht diese verloren.
- parallele Bedienstation ***c*** – Anzahl paralleler Bedienstationen.
- Bedienzeitverteilung ***B*** – beschreibt wie die Entitäten an der Bedienstation bearbeitet werden. Die Bedienzeitverteilung kann man mit einer Verteilungsfunktion beschrieben werden. Ein Beispiel wäre die Normalverteilung.

Zur Unterscheidung der verschiedenen Warteschlangenmodelle nutzt man dann die Abkürzung ***A/B/c/N/K*** . Die Parameter ***A*** und ***B*** haben dabei folgende Wertebereiche:

- ***M*** = exponentiell (markovian)
- ***D*** = deterministisch (deterministic)
- ***G*** = beliebig (general)

Hinweis: Ist die Warteschlangengröße und die Kundenpopulation unendlich groß, so lassen wir die letzten beiden Parameter weg: $A/B/c/\infty/\infty \Leftrightarrow A/B/c$.

Wichtige Ergebnisse

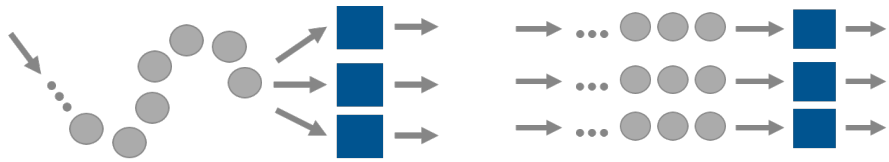
Möchte man nun das Modell auswerten sind folgende Werte von besonderer Bedeutung:

- $\lambda = E[A]$ – mittlere Ankunftsrate
- $\mu = E[B]$ – (mittlere) Bedienrate pro Station
- $\sigma^2 = Var[B]$ – Streuung der Bedienrate

Ist die Ankunftsrate kleiner als die Bedienrate pro Station, können also die Entitäten schneller bearbeitet werden als diese ankommen, spricht man von*Stationarität*. Eine weitere Feststellung, die aus der *Stationarität* folgt ist, dass die Warteschlange nicht ins unendliche wächst. Weitere Größen sind:

- $L(t)$ – Gesamtzahl der Kunden im System zur Zeit t
- $L(t) \rightarrow L(\infty)$ – stationäre Verteilung
- L – Erwartungswert von $L(\infty)$
- ρ – Langzeitausnutzung der Server
- w – Langzeitaufenthalt im System / Mittlere Aufenthaltszeit der Aufträge

Ziel der Warteschlangentheorie ist es dann die Verteilung von $L(\infty)$ zu bestimmen. Ergebnisse für bestimmte Warteschlangen sind dann hier[1] zu finden. Zu guter letzt können wir festhalten, dass bei konstantem Erwartungswert und wachsender Streuung der Bedienzeit Warteschlangen länger werden. Außerdem tritt jede Warteschlangenlänge mit gewisser Wahrscheinlichkeit auf, d.h. Warteschlangen laufen stets gelegentlich voll und *n*Bedienstationen mit einer Warteschlange sind besser als *n* Bedienstationen mit *n* Warteschlangen.



Gesetz von Little

Das Gesetz von Little wird zur Konsistenzprüfung von Simulationsergebnissen verwendet. Es ist wichtig zu beachten, dass das Gesetz von Little auf alle stationären Warteschlangenmodelle angewendet werden kann. Es ist also unabhängig von der Ankunfts- und Bedienverteilung! Es besagt, die mittlere Anzahl der Aufträge entspricht dem Produkt von Eintreffrate und Aufenthaltszeit.

- $L = \lambda \cdot w$

- L – mittlere Zahl der Aufträge im System
- λ – mittlere Eintreffrate der Aufträge
- w – mittlere Aufenthaltszeit der Aufträge

Beispiel

Man betrachte das Wartenschlangenmodell der Tankstelle aus der Vorlesung. Man nehme an es kommen vier Kunden pro Minute mit exponentialverteilten Ankunftszeiten. Die Bezahlung dauert durchschnittlich 12 Sekunden und ist ebenfalls exponentialverteilt. Die einzige Möglichkeit zu bezahlen ist der Automat. Damit ergibt sich eine ***M/M/1*** - Warteschlange. Die Formeln dazu findet man hier[1]. Die Bedingung für *Stationarität* wäre $\lambda < \mu$. Mit vier Kunden pro Minute wäre $\lambda = 4$. Da das Bezahlen am Automaten im Durchschnitt 12 Sekunden dauert kommt man für μ auf $\mu = 60/12 = 5$. Es können pro Minute also ungefähr 5 Kunden an der Bedienstation bearbeitet werden. Damit gilt für diese Warteschlange*Stationarität*. Für die Langzeitausnutzung folgt $\rho = \lambda/\mu = 4/5 = 0.8$. Wir haben also eine 80% Auslastung der Bedienstation. Der Erwartungswert der Anzahl der Kunden im System wäre dann $L = \rho/(1-\rho) = 0.8/(1-0.8) = 4$ und die Langzeitaufenthaltszeit beträgt $w = 1/(\mu - \lambda) = 1/(5-4) = 1$.

Einführung in Computational Engineering – Vorlesung 4

Autoren: Andrej Felde und Thomas Hesse

Örtliche konzentrierte und verteilte Parameter

Wir wollen uns zuerst eine Intuition für die Begriffe des örtlich konzentrierten und verteilten Parameters verschaffen. Im allgemeinem verstehen wir diese Begrifflichkeiten als Differentialgleichungen. Den **örtlich konzentrierten Parameter** verstehen wir dabei als ein Differential von einem Parameter. Das bedeutet im Umkehrschluss, dass wir hierbei von gewöhnlichen Differentialgleichungen reden. Bei dem **örtlich verteilten Parameter**reden wir dann von partiellen Differentialgleichung. Dies lässt sich im Allgemeinem als eine Funktion mit mehreren Parametern betrachten. Beispiele dafür haben wir in der Vorlesung gesehen.

Jacobi Matrix

Bei der Jacobi Matrix handelt es sich um eine Methode Funktionen mehrerer Parameter zu differenzieren. Diese Methodik sollten sie schon in der Mathematik für Informatik 2 angeeignet haben. Wir wiederholen sie hier dennoch noch einmal kurz der Vollständigkeit wegen. Außerdem werden wir zeigen, wie man die Jacobi Matrix analytisch und numerisch herleitet.

Analytische Herleitung der Jacobi Matrix

Möchte man die Jacobi Matrix J einer Funktion $f(a)$ bestimmen, dann macht man dies mittels:

$$J_f(a) := \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(a) & \frac{\partial f_1}{\partial x_2}(a) & \dots & \frac{\partial f_1}{\partial x_n}(a) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1}(a) & \frac{\partial f_m}{\partial x_2}(a) & \dots & \frac{\partial f_m}{\partial x_n}(a) \end{pmatrix}.$$

Numerische Herleitung der Jacobi Matrix

Die Jacobi Matrix kann auch numerisch über den Vorwärtsdifferenzenquotienten berechnet werden. Es gilt:
$J = \frac{\partial f}{\partial x_i} = \frac{1}{\delta_j} \cdot (f(x + e_j \cdot \delta_j) - f(x))$

Dabei steht e_j für den j-ten Einheitsvektor, δ_j steht für die Schrittweite bezüglich des j-ten Systemzustandes zusammen mit einer persönlich festgelegten Toleranzgrenze ε. Wir haben hierbei schon in der Vorlesung gesehen, wie man sich z.B. Werte wie δ_j wählen kann (dies ist nicht-trivial!). Des Weiteren wird in den Folien eine Intuition gegeben, wie diese Schrittweite zu verstehen ist.

Transformation einer Differentialgleichung (DGL)

Angenommen wir haben eine DGL n-*ter* Ordnung, dann können wir diese in ein DGL-System der Dimension n transformieren. Dafür führen wir den Systemzustand x ein, wobei die i-*te* Komponente des Systemzustandvektors den ursprünglichen Systemzustand vom Grad i enthält. Wir betrachten dieses Vorgehen an folgendem Sachverhalt: Angenommen wir haben ein DGL System zweiter Ordnung von der Form $\ddot{x} + 5\dot{x} + 2x = 0$. Als erstes substituieren wir die Systemzustände. Sei $z_1 = x(t)$ und $z_2 = \dot{x}(t)$. Diese bilden dann den Vektor z. Dann können wir auch schon das DGL System aufstellen! Es ergibt sich:

$$\dot{z} = \begin{pmatrix} \dot{z}_1 \\ \dot{z}_2 \end{pmatrix} = \begin{pmatrix} z_2 \\ -5z_2 - 2z_1 \end{pmatrix}$$

Damit haben wir dann eine DGL zweiter Ordnung auf ein DGL System erster Ordnung transformiert.

Autonomisierung einer Differentialgleichung (DGL)

Bei der Autonomisierung von Differentialgleichungen geht es lediglich darum, die Differentialgleichung nicht mehr explizit von der Zeit abhängig zu machen. Dies hat den Vorteil, dass das System dann nur noch von Systemzuständen abhängt und nicht mehr von einer zeitlichen Komponente.

Lineare und nicht-lineare Systeme

Bei Differentialgleichungen können zwischen linearen und nicht-linearen Systemen differenziert werden.

Lineare Systeme

Bei linearen Systemen handelt es sich meist um sehr einfache Differentialgleichungen der Form $\dot{x} = A \cdot x + B \cdot u$ vereinfacht dargestellt. Hierbei handelt es sich bei der Matrix A um die Systemmatrix und bei B um die Zustandsmatrix. Lineare Systeme kann man sich wie eine Funktion $f(x) = x$ vorstellen, wobei A die darstellende Matrix der Funktion $f(x)$ wäre.

Nicht-lineare Systeme

Nicht-lineare Systeme sind im Gegensatz zu den linearen Systemen meist sehr komplexe Differentialgleichungen der Form $\dot{x} = f(x, u)$vereinfacht dargestellt. Bei der Funktion $f(x, u)$handelt es sich dabei um eine nicht-lineare Funktion. Diese können sich z.B. aus verschiedenen interpolierten Polynomen zusammensetzen oder gar Splines sein. Für solche Problemstellung stellt die Mathematik zudem nicht so viele Rechenregeln zur Verfügung. Oft ist der Ansatz, dass man nicht-lineare Systeme einfach in lineare Systeme überführt. Dafür benutzen wir die Linearisierung in einem Punkt.

Existenz einer Lösung zu einer Differentialgleichung (DGL)

Die Lösung einer linearen Differentialgleichung existiert genau dann, wenn wir eine reguläre Systemmatrix vorliegen haben. Das heißt, dass die Matrix insbesondere invertierbar ist und damit auch vollen Rang hat. Die Existenz einer Lösung von einer nicht-linearen Differentialgleichung lässt sich so z.B. nachweisen mit dem Satz von Picard-Lindelöf, welcher uns dann auch später zur Fixpunkt Iteration (kurz: FPI) führt! Eine weitere Alternative zur Feststellung der Existenz einer Lösung zu einer nicht-linearen Differentialgleichung bietet der Satz von Peano.

Linearisierung um Gleichgewichtslage / Gleichgewichtspunkt

Bei der Linearisierung um einen Punkt handelt es sich im allgemeinem um eine tangentiale Approximation an eine

nicht-lineare Funktion mit einer linearen Funktion, wobei nur dieser eine Punkt exakt dargestellt ist. Man berechnet diese wie folgt:

1.Bestimmung der allgemeinen Jacobi Matrix der nicht-linearen Funktion $f(x,y)$, sodass man $J_f(x,y)$ erhält.

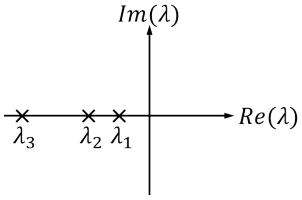
2.Einsetzen des Punktes / Gleichgewichtspunktes um den linearisiert werden soll, also $J_f(x_s,y_s)$.

3.Dann lässt sich die linearisierte Funktion um diesen Punkt mit $\dot{\Delta x} = J_f(x_s,y_s)\cdot\Delta x$ aufstellen. Diese Betrachtung folgt aus der Herleitung durch die Taylorreihe oder des Taylorpolynoms für eine gewisse $\mathcal{E}$-Genauigkeit.

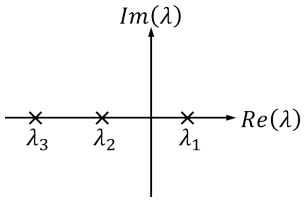
Natürlich ist dieser Vorgang nur für Punkte $(x,y)\in\mathbb{R}^2$ zulässig, ist aber genau so wie er dort steht auf ein anders-dimensionales Beispiel zu übertragen.

Stabilitätsbetrachtung von Systemen

Bei der Stabilitätsbetrachtung von Systemen setzen wir die Berechnung von Eigenwerte voraus. Ein möglicher analytischer Ansatz zum Berechnen von Eigenwerten wäre demnach zum Beispiel mithilfe des charakteristischen Polynoms $\det(A - I\cdot\lambda)$ mit Systemmatrix A. Nun wollen wir im folgenden anhand der Eigenwerte eines Systems die Stabilität im reellen, sowie im imaginären bestimmen können. Systeme, bei denen sich die Eigenwerte im imaginärem Bereich bewegen, basieren meistens auf Amplituden und Schwingungen. Wir betrachten im folgendem die Fälle von Stabilität und Instabilität für reelle Eigenwerte:

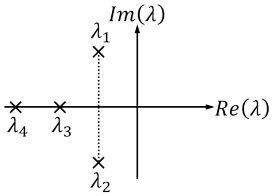


•Die Eigenwerte sind hier wie gesagt alle reell und alle kleiner null. Damit ist diese Grafik ein Beispiel für ein stabiles System.

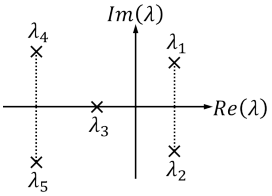


•Die Eigenwerte bei diesem System sind nicht alle kleiner null. Damit ist das System im Gesamten instabil.

Nun betrachten wir Stabilität und Instabilität für komplexe Eigenwerte:



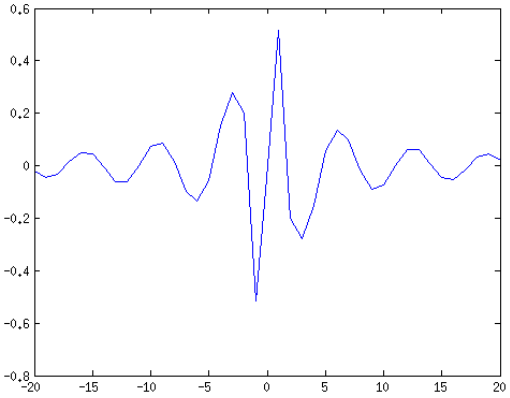
•Die Eigenwerte sind hier wie gesagt alle komplex und im Realteil alle kleiner null. Damit ist diese Grafik ein Beispiel für ein stabiles System.



•Die Eigenwerte bei diesem System sind für den Realteil einiger Eigenwerte nicht kleiner null. Damit ist das System im Gesamten instabil.

Einführung in Computational Engineering – Vorlesung 5

Autoren: Andrej Felde und Thomas Hesse



Wir haben in der Vorlesung das Beispiel zu dem inversen Pendel gesehen! Nun wird der bisher gelernte Stoff anhand einer einfachen gewöhnlichen Differentialgleichung wiederholt. Sei dafür die Differentialgleichung $\dot{x} = \sin(\cos(x))$. Dieses System hat bei $x(0)$ eine Sprungstelle und ist dort nicht definiert. Sei dafür $x(0) = 0$. Die durch die Systemmatrix beschriebene Funktion ist links in dem Plot abgebildet. Im Folgendem werden wir nur noch diese Differentialgleichung betrachten.

Stabilität

Im folgendem wird der Eigenwert der Systemmatrix berechnet. Die Systemmatrix ist jedoch ein skalar, man kann diese also als eindimensionale Diagonalmatrix verallgemeinern und sofort den einzigen Eigenwert des System ablesen. Dieser ist mit $\lambda = \frac{\sin(\cos(x))}{x}$ gegeben. Offensichtlich existiert ein x, für das gilt $\mathrm{Re}\{\lambda\} > 0$, sodass dieses System **instabil** ist.

Steifheit

Das System ist nicht steif, da $T_1 = T_{max} = T_{min}$ und damit für das Steifheitsmaß $s = 1$ gilt und $1 < 10^3 \dots 10^7$ ist. Man beachte, dass solch ein Fall durchaus auch für Differentialgleichungssysteme auftreten kann und damit nicht nur auf Differentialgleichungen 1-ter Ordnung beschränkt ist!

Linearisierung um einen Punkt

Zuerst bestimme man das lineare System um den Arbeitspunkt x_0. Dazu stelle man zuerst die Jakobi Matrix des Systems auf. Das ist in diesem Falle der Gradient $\nabla x = -\cos(\cos(x))\sin(x)$! Damit ergibt sich das System $\Delta\dot{x} = \nabla x \Delta x$.

Linearisierung um die Gleichgewichtslage(n)

Die Gleichgewichtslagen befinden sich bei $x_s = \arccos(n\pi)$ für $n \in \mathbb{Z}$. Einsetzen in den Gradienten ergibt

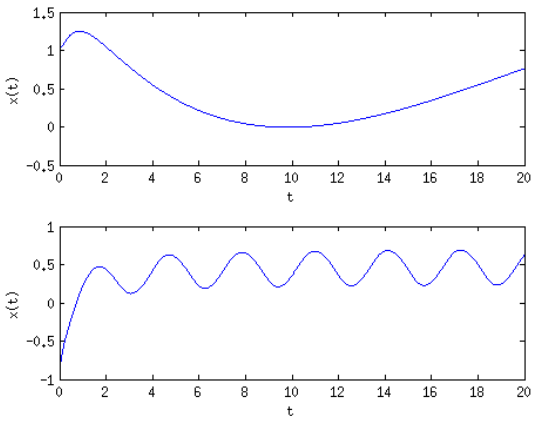
$$\nabla x = -\sqrt{1 - n^2\pi^2}\cos(n\pi) = -\sqrt{1 - n^2\pi^2}(-1)^n$$

und damit ergibt sich auch gleich das linearisierte System um die Gleichgewichtslage

$$\Delta\dot{x} = \left(-\sqrt{1 - n^2\pi^2}(-1)^n\right)\Delta x.$$

Linearisierung um Referenztrajektorie

Bei der Linearisierung um die Referenztrajektorie betrachten wir ein System über einen zeitlichen Verlauf von Zuständen hinweg. Dies ist gerade die allgemeine Linearisierung um einen Punkt $\Delta\dot{x}(t) = \nabla x \Delta x(t)$, wobei wir die Trajektorie als zeitliche Abfolge von Systemzuständen betrachten. Folgend noch Beispiele für Referenztrajektorien und wie man sich diese vorstellen kann.



Einführung in Computational Engineering – Vorlesung 6

Autoren: Andrej Felde und Thomas Hesse

Zahlendarstellung

Reelle Zahlen werden auf Computern als normalisierte Gleitpunktzahlen dargestellt. Eine normalisierte Gleitpunktzahl wird allgemein definiert durch:

$$z = \pm(d_1 \cdot B^0 + d_2 \cdot B^{-1} + \cdots + d_t \cdot B^{t-1}) \cdot B^E = \sum_{i=1}^{t} \left(d_i \cdot B^{-i+1}\right) \cdot B^E$$

Dabei sind die Zahlen wie folgt definiert:

•Basis B (Dezimalsystem $B = 10$ und im Binärsystem $B = 2$)

•Exponent E ist eine ganze Zahl und ist durch $E_{min} \leq E \leq E_{max}$ beschränkt. (Bei $3.14 \cdot 10^1$ wäre $E = 1$)

•Ziffern $d_i \in \{0, 1, \ldots, B-1\}$ stellen die Zahlen der Zahl dar. (Bei $3.14 \cdot 10^1$ wäre $d_1 = 3, d_2 = 1$ und $d_3 = 4$

•Die Länge der Mantisse wird durch t beschrieben. (Bei $3.14 \cdot 10^1$ wäre $t = 3$)

•Die Mantisse selbst besteht aus den Ziffern d_i

Man kann sich die Darstellung anhand einer normalisierten Dezimalzahl verdeutlichen. Nur die erste Stelle vor dem Dezimalpunkt ist ungleich Null, wenn die Zahl selbst nicht die Null ist. Alle anderen Stellen vor dem Dezimalpunkt (Dezimalkomma) sind gleich Null. Auf heutigen Computern werden reelle Zahlen allerdings als Binärzahlen dargestellt. Zudem besteht die gängige Speicherwortlänge aus 32 Bit. Damit ergibt sich eine Aufteilung von einem Bit für das Vorzeichen, 8 Bit für den Exponenten und 23 Bit für die Mantisse selbst. Der Wert der Zahl kann dann mit $(-1)^S \times (1 + M) \times 2^E$

Stellt man mehr Bits für den Exponenten E bereit ergibt sich ein größerer Bereich der dargestellt werden kann. Stellt man hingegen mehr Bits für die Mantisse M bereit, so können die Zahlen genauer dargestellt werden. Zur Standardisierung wird der IEEE 754 Gleitpunktstandard verwendet.

IEEE 754 Gleitpunktstandard

Beim Gleitpunktstandard wird festgelegt wie viele Bit für den Exponenten bzw. für die Mantisse bereitgestellt werden. Bei einfacher Genauigkeit (single precision) hat der Exponent 8 Bit und die Mantisse 23 Bit. Bei doppelter Genauigkeit (double precision) hat der Exponent 11 Bit und die Mantisse 52 Bit. Außerdem wird das oberste Bit der Mantisse also d_1 weggelassen, da dieses implizit immer da ist, wenn die Zahl selbst ungleich Null ist. Zudem wird der Bias eingeführt. Der Bias dient zur Verhinderung von negativen Exponenten sowie zur Verhinderung von zusätzlichen Aufwand bei der Darstellung. Dazu wird zu einem Exponenten e der Bias B addiert. Dieser Wert $E = e + B$ wird dann gespeichert. Bei einfacher Genauigkeit ist der Bias $B = 127$ und bei doppelter Genauigkeit ist der Bias $B = 1023$.

Sonderfälle

Ist die Zahl selbst die Null wird diese als reservierter Sonderfall behandelt. Dabei sind dann alle 32-Bit (bei single precision) gleich 0. Ein Überblick über die Werte kann man auch folgender Tabelle entnehmen.

Exponent	Mantisse	dargestelltes Objekt
00...00	00...00	0
1–254	beliebig	± normalisierte Gleitpunktzahl
00...00	beliebig, von Null verschieden	± sub-normale Zahl (ohne Exponent)
255	00...00	± ∞ Unendlich (infinity)
255	beliebig, von Null verschieden	NaN (not a number)

Eigenschaften

Zusammenfassend kann man folgende Eigenschaften aufzählen
•Es gibt nur endlich viele Gleitpunktzahlen

•Es gibt keine beliebig großen und keine beliebig kleinen (positiven) Gleitpunktzahlen

•Es gibt keine beliebig nahe benachbarte Zahlen

•Die Gleitpunktzahlen sind ungleichmäßig verteilt

•Summe/Differenz, Produkt/Quotient von Gleitpunktzahlen müssen i.Allg. gerundet werden

Rundungsfehler

Wie im vorherigen Kapitel bereits gezeigt wurde gibt es nur endlich viele Gleitpunktzahlen und nach Summe / Differenz, Produkt / Quotient muss im Allgemeinen gerundet werden. Rundungsfehler sind jedem ein Begriff, allerdings ist die Frage wie sich Rundungsfehler auswirken. Allgemein lässt sich Runden wie folgt definieren:
$|x - rd(x)| \leq |x - g|, \quad \forall g.$

Dabei steht g für eine Gleitpunktzahl. Da das Ergebnis einer arithmetischen Operation im Allgemeinen keine Maschinenzahl ist, muss das Ergebnis gerundet werden. Beim Runden kann man den relativen Rundungsfehler bestimmen. Die relative Maschinengenauigkeit ε_{mach} ist zum Beispiel auch nichts anderes als der maximale relative Rundungsfehler. Zudem kann man auch den absoluten Rundungsfehler bestimmen. Dieser gibt einfach nur die Abweichung der gerundeten Gleitkommazahl zur reellen Zahl an. Den relativen Rundungsfehler bestimmt man durch:
$\varepsilon(x) = \frac{x - rd(x)}{x}$

Es kann festgehalten werden, dass für Gleitpunktarithmetik weder das Assoziativ- noch das Distributivgesetz gelten. Dies kann man sich durch die Addition von zwei sehr kleinen und ähnlichen Zahlen x,y mit einer sehr großen Zahl z verdeutlichen. Addiert man nun zuerst x und y kommt eine potenziell größere Zahl raus. Diese hat dann, da sie näher an y liegt, auch einen größeren Einfluss auf das Ergebnis. Addiert man nun x oder y zuerst mit z hat die sehr kleine Zahl kaum Einfluss auf das Ergebnis. Dabei wird durch Runden und durch die beschränkte Größe der Mantisse das Zwischenergebnis kaum geändert. Das Gleiche passiert dann bei der Addition der zweiten kleinen Zahl.
Ein kleines Beispiel wie sich Rundungsfehler auswirken können sieht man auch noch in folgendem Beispiel, welches bereits aus der Vorlesung bekannt sein sollte. Weitere Bespiele kann man auch noch hier [1] finden.

Auslöschung

Die Auslöschung beschreibt den Verlust von Bits, die beim Speichern einer sehr großen oder sehr kleinen Zahl auftreten. Auslöschung kann durchaus auch bei Gleitpunktarithmetischen Operationen auftreten. Als Beispiel nehme man Zahlen an, die durch eine Reihe approximativ dargestellt werden, so in etwa π.

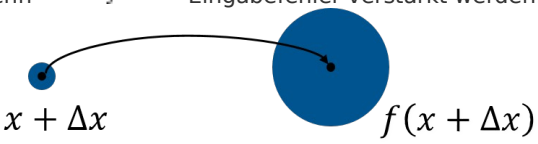
Kondition

Die Kondition beschreibt wie sich Rundungsfehler auf eine Funktion $f(x)$ auswirken. Dabei versteht man unter "schlecht konditioniert", dass kleine Änderungen in x große Änderungen in $f(x)$ bewirken. Die Konditionszahlen sind die Beträge der Verstärkungsfaktoren zum relativen Fehler von y. Der relative Fehler ist mit Taylorentwicklung gegeben durch:
$\varepsilon_{y_i} \approx \sum_{j=1}^{n} \left(\frac{x_j}{f_i(x)} \cdot \frac{\partial f_i(x)}{\partial x_j} \right) \cdot \varepsilon_{x_j}$

Damit sind die Konditionszahlen gegeben durch:

$$cond_f = \left| \frac{x_j}{f_i(x)} \cdot \frac{\partial f_i(x)}{\partial x_j} \right|$$

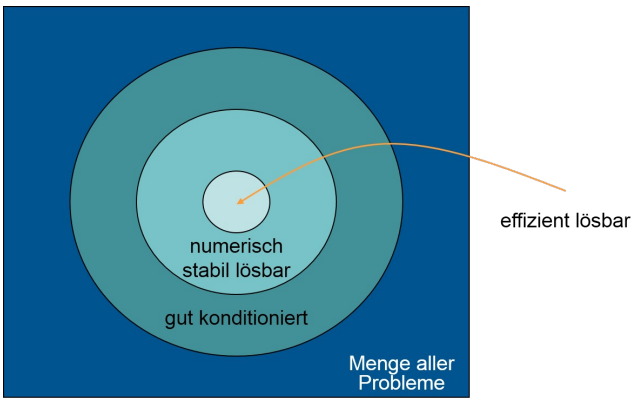
Sind die Konditionszahlen "*groß*", ist das Problem "schlecht konditioniert". Sind die Konditionszahlen $cond_f$ hingegen "*klein*", ist das Problem "gut konditioniert". Das Besondere an den Konditionszahlen ist, dass diese nur von den Eigenschaften der Funktion f abhängen. Es besteht keine Abhängigkeit zur Auswertungsart oder wie die Rechnerarithmetik aussieht!
Entsprechend kann man sich vorstellen , dass wenn $cond_f \leq 1$ Eingabefehler nicht verstärkt werden und wenn $cond_f > 1$ Eingabefehler verstärkt werden.



Numerische Stabilität

Die numerische Stabilität ist nochmal eine Verschärfung zur Kondition. Unter der numerischen Stabilität versteht man wie Eingabefehler das Ergebnis in Hinblick auf die Auswertung beeinflussen, selbst wenn das Problem gut konditioniert ist.
•Ein Berechnungsverfahren für f nennt man numerisch stabil, falls die relevanten Eingabefehler nicht verstärkt werden

•Ein Berechnungsverfahren, nennt man numerisch instabil, falls große relative Fehler im Ergebnis erzeugt werden

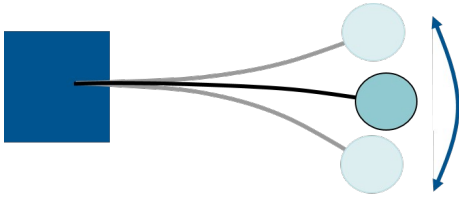


Man sieht, dass gut konditionierte Probleme starke Rundungsfehler aufweisen können, wenn sie numerisch instabil sind. Bei der numerischen Stabilität gibt man also eine Aussage über die Ergebnisqualität nach der Auswertung durch einen Rechner!

Einführung in Computational Engineering – Vorlesung 7

Autoren: Andrej Felde und Thomas Hesse

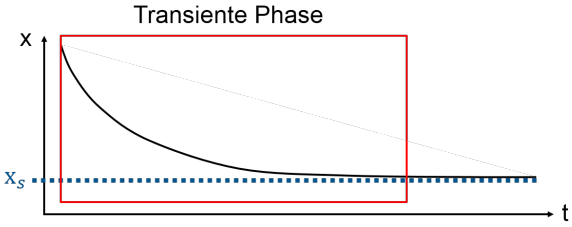
Differentialgleichungslöser



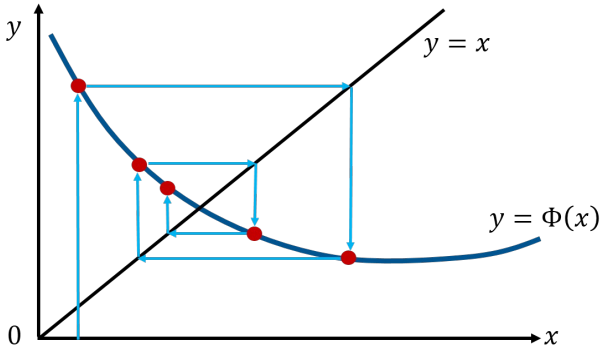
Oft hat man es in der Praxis mit nichtlinearen Problemen zu tun. Hier wird nun versucht, eine effiziente Möglichkeit zu erarbeiten, mit der man die Gleichgewichtslösung einer nicht lineare Differentialgleichung bestimmen kann. Die Lösung dazu lautet "Differentialgleichungslöser". Mit den Differentialgleichungslösern versucht man im Allgemeinem stationäre Punkte oder eben nur einen stationären Punkt einer Differentialgleichung zu bestimmen. Dafür werden in der Regel Kontraktionen benutzt (selbst-abbildende Abbildungen). Im folgendem werden zwei Verfahren vorgestellt, welche den stationären Punkt (also die Gleichgewichtslage) einer Differentialgleichung bestimmen. Ein Beispiel für das zu bestimmende stationäre System sähe wie folgt aus:

Transiente Phase

Die transiente Phase beschreibt genau den Zeitbereich, in dem das nicht lineare System sich nicht auf die Gleichgewichtslage einpendelt. Interessant ist demnach natürlich der Zustand des Systems nach der transienten Phase. Die Verfahren, welche zur Bestimmung der Gleichgewichtslage benutzt werden, versuchen gerade diese transiente Phase mit iterativen Ansätzen bis hin zur Gleichgewichtslösung x_s zu durchlaufen.



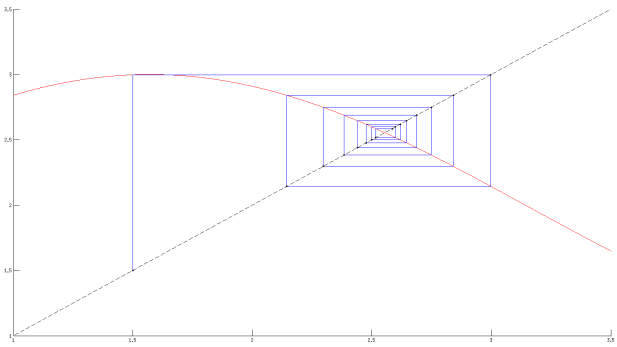
Fixpunktiteration



Allgemein handelt es sich bei der Fixpunktiteration um eine Abbildung, genauer eine Kontraktion. Eine Kontraktion ist eine Abbildung, die von sich wieder auf sich selbst abbildet, also eine selbst abbildende Abbildung. Für den Falle der Fixpunktiteration wird konkret von einem Banachraum nach einem Banachraum abgebildet. Ein Banachraum ist ein Vektorraum, auf dem eine Metrik (z.B. die euklidische Norm) definiert wurde und der vollständig ist. Das "vollständigkeits"-Kriterium ist genau dann erfüllt, wenn jeder Punkt des Raumes Cauchy-konvergent ist.

Die in der Vorlesung vorgestellte Form für die Fixpunktiteration einer nicht linearen Differentialgleichung $f(x) = \dot{x} = x$ wäre demnach $\Phi(x) = x$ mit der Kontraktion $\Phi(x)$, da ja gerade $f(x) \overset{!}{=} 0$ gelten muss!

Beispiel



Man betrachte die Fixpunktiteration für die Kontraktion $\Phi(x) = sin(x) + 2$ mit dem Startwert $x(0) = 1.5$. Die ersten 5 Iterationen sind wie folgt:

1. $\Phi(1.5) = 2.997495$
2. $\Phi(2.997495) = 2.1436$
3. $\Phi(2.1436) = 2.840385$
4. $\Phi(2.840385) = 2.296674$
5. $\Phi(2.296674) = 2.747917$

[...] Führt man nun dieses Verfahren weiter fort, so erhält man zum Schluss den Fixpunkt $x_s \approx 2.55$, welcher die Gleichgewichtslage an der ungefähren x-Position von ebenfalls 2.55 ergibt. Das Bild der Fixpunktiteration mit den ersten 15 Iteration kann man sich hier noch einmal genauer anschauen.

Newton-Verfahren

Das Newton-Verfahren ist ein spezieller Fall der Fixpunktiteration genau dann, wenn die Folge gegen den Fixpunkt x_s konvergiert. Das Newton-Verfahren wird ebenso wie die Linearisierung um die Gleichgewichtslage [1] mit dem Taylorpolynom ersten Grades hergeleitet, also bis zum linearen Term. Das Newton-Verfahren macht nichts

anderes, als an bestimmten Funktionswerten zu linearisieren und den Wert an dem sich die x-Achse mit der Tangente schneidet als neuen Funktionswert zu verwenden. Hier zeichnet sich die iterative Vorgehensweise des Newton-Verfahrens ab. Die allgemeine Form der Newton Gleichung ist:
$x_{i+1} = x_i + \Delta x_i \qquad \text{mit} \qquad \frac{\partial f}{\partial x}(x_i) \cdot \Delta x_i = -f(x_i)$

Ein Nachteil dieses Verfahrens ist die lokale Konvergenz. Durch die lokale Konvergenz kann es sein, dass das Verfahren zu große Schritte macht und deshalb divergiert. Das Newton-Verfahren wird solange ausgeführt, bis es terminiert. Die zwei Terminierungskritierien sind:
1.Nahe der Lösung – in diesem Falle erreicht das Newton-Verfahren die Gleichgewichtslösung oder befindet sich in ε -Nähe zu der Gleichgewichtslösung

2.Kein Fortschritt – festgelegte "Fortschrittsgrenze" ε wird zwischen den Iterationen unterschritten, ansonsten kann man ein Maximum an Iterationen festlegen

Einführung in Computational Engineering – Vorlesung 8

Autor: Hong Linh Thai und Johannes Alef

(Quasi-)Newton-Verfahren:

Das Quasinewton-Verfahren baut auf dem Newton-Verfahren [1] auf und verwendet den gleichen Ansatz wie das Newton-Verfahren mit:
$x_{i+1} = x_i + \Delta x_i \qquad \text{mit} \qquad \frac{\partial f}{\partial x}(x_i) \cdot \Delta x_i = -f(x_i)$

Jedoch wird die Jacobi Matrix in dem Fall nicht *analytisch* bestimmt, sondern *numerisch*. Dazu gibt es folgende Möglichkeiten :
•Approximation der von Null verschiedenen Einträge mit Differenzenquotienten, z.b. Vorwärts-Differenzenquotienten[2]

•Man ersetzt die Jacobi Matrix durch eine vorbestimmte Lösung z.b. $\frac{\partial f}{\partial x}(x_i)$ durch $\frac{\partial f}{\partial x}(x_0)$

•Man approximiert die Jacobi Matrix schrittweise in jeder Iteration z.b. Broyden's rank 1 update

Es ergibt sich folgende Rechenvorschrift :
Gegeben : $x_0, f : \mathbb{R}^n \to \mathbb{R}^n$
•Berechne $f(x_i)$

•Berechne die Näherung für die Jacobi-Matrix: J_i

•Löse das Gleichungssystem $J_i \cdot \Delta x_i = -f(x_i)$ nach Δx_i auf.

•Berechne $x_{i+1} = x_i + \Delta x_i$ bzw. bei Nutzung einer Schrittweitensteuerung berechne $x_{i+1} = x_i + \alpha_i \cdot \Delta x_i$

Im Falle der Berechnung von J_i durch Aufdatierung gilt: $J_i = J_{i-1} + U_i$.
Beispielhafte Aufdatierung: Broyden's Rang 1 Update:
•Beginne mit $J_0 = I$ (Einheitsmatrix).
• $U_i = \frac{1}{\|\alpha_{i-1} \cdot \Delta x_{i-1}\|_2^2} \cdot [f(x_i) - f(x_{i-1}) - J_{i-1} \cdot \alpha_{i-1} \cdot \Delta x_{i-1}] \cdot [\alpha_{i-1} \cdot \Delta x_{i-1}]^T$

Numerische Integration:

Idee:

Für nichtlineare DGLs ist es im Allgemeinen nicht möglich, eine analytische Lösung zu nutzen. Diese DGLs werden deswegen im Rahmen von Simulationen numerisch gelöst.
Der Grundgedanke ist hierbei die Lösung zu diskreten Zeitpunkten zu betrachten. Betractet man die DGL $\dot{x} = f(x)$. Nun will man die Lösung zu Zeitpunkten $t_{i+1} = t_i + h_i$.

Mit den Rechenregeln für Integrale ergibt sich
$x(t_{i+1}) = x(0) + \int_0^{t_i} f(x(\tau))d\tau + \int_{t_i}^{t_{i+1}} f(x(\tau))d\tau = x(t_{i+1}) + \int_{t_i}^{t_{i+1}} f(x(\tau))d\tau$
Damit kann als Ausgangsgleichung für die Diskretisierung nun die folgende Gleichung genutzt werden:
$x(t_{i+1}) = x(t_{i+1}) + \int_{t_i}^{t_{i+1}} f(x(\tau))d\tau$.

Diese lautet in diskreter Form:
$x_{i+1} = x_i + h_i \cdot f_i$.

Dabei gilt $f_i = f(x(t_i))$
Auf diese Weise lässt sich die Integralfläche z.B. als Rechteck unter der Funktion approximieren.

Übersicht:

Integrationsverfahren lassen sich in mehrere Arten unterteilen anhand der Art, wie die Fläche der Funktion approximiert wird und wie der Gradient approximiert wird.
•Einschrittverfahren

Einschrittverfahren betrachten nur einen Integrationsschritt, also von x_i zu x_{i+1}.
•Mehrschrittverfahren

Mehrschrittverfahren betrachten mehrere Integrationsschritte.
•Extrapolationsverfahren

Diese Verfahren lassen sich nochmals unterteilen in
•explizite Verfahren

•implizite Verfahren

Im Rahmen dieser Vorlesung werden nur Einschrittverfahren betrachtet. Für diese gilt generell:
Gegeben ist x_i. Dies entspricht ungefähr dem exakten Wert $x(t_i)$.
Gesucht wird der Wert x_{i+1}, der ungefähr dem exakten Wert $x(t_{i+1})$ entspricht.
Der allgemeine Ansatz lautet: $x_{i+1} = x_i + h \cdot \Phi(t_i, x_i, x_{i+1}, h; f)$. h bezeichnet dabei die verwendete Schrittweite, t_i den aktuellen Zeitpunkt.
Dabei ist die Funktion Φ vom verwendeten Einschrittverfahren abhängig.

Die Verfahren müssen folgende Konvergenzbedingung erfüllen:
$lim_{h->0}\Phi(t_i, x_i, x_{i+1}, h; f) = f(x_i)$

Das heißt, je kleiner man die Schrittweite wählt, desto besser muss der approximierte Wert den exakten Wert annähern.

Expliziter Euler:

Das explizite Euler-Verfahren ist eines der einfachsten Einschrittverfahren. Hier wird $\Phi(t_i, x_i, x_{i+1}, h; f) = \Phi(t_i, x_i) = f(x_i)$ gewählt. Damit ergibt sich als Rechenvorschrift $x_{i+1} = x_i + h \cdot f(x_i)$.
Der Rechenaufwand für dieses Verfahren ist sehr gering, da f nur einmal pro Integrationsschritt ausgewertet werden muss.

Fehler:

Bei der Verwendung von Einzelschrittverfahren entsteht bei jedem Schritt ein sogenannter Einzelschrittfehler. Dieser beschreibt die Abweichung Der Approximation von der exakten Lösung nach einem Schritt, wenn man davon ausgeht, dass der Ausgangswert exakt ist.Zusätzlich entsteht noch ein Fortpflanzungsfehler dadurch, dass die Startwerte für die Iterationsschritte schon ungenau sind und dadurch die Funktion verfälscht ist. Diese beiden Fehler lassen durch kleinere Schrittweiten minimieren. Je kleiner jedoch die Schrittweite ist, desto größer wird der Einfluss von Rundungsfehlern.
Um den Gesamtfehler klein zu halten und trotzdem auch hohe Schrittweiten nutzen zu können kann man eine adaptive Schrittweitensteuerung nutzen. Mit dieser können in Bereichen der Funktion, in denen kleine Schritte

notwendig sind kleine Schritte gemacht werden, in wenig veränderlichen Bereichen aber große Schritte.

Impliziter Euler:

Beim impliziten Euler-Verfahren wird $\Phi(t_i, x_i, x_{i+1}, h; f) = \Phi(x_{i+1}) = f(x_{i+1})$ gewählt. Der Rechenaufwand ist hier sehr viel höher als beim expliziten Euler-Verfahren, da man eine nichtlineare Gleichung lösen muss. Der Fehler liegt auch hier in $O(h^1)$. Für eine hohe Genauigkeit ist sind also kleine Schrittweiten nötig. Im Gegensatz zu expliziten Verfahren sind implizite Verfahren aber wesentlich stabiler bei steifen Differentielgleichungen.
Diese Verfahren lassen sich auch auf Funktionen mit Dimension größer 1 verallgemeinern. Im mehrdimensionalen Fall werden die Komponenten einzeln berechnet.

Einführung in Computational Engineering – Vorlesung 9

Autor: Johannes Alef

Neben den Euler-Verfahren gibt es noch weitere Verfahren zur numerischen Integration. Im folgenden sollen zwei Verfahren höherer Ordnung gezeigt werden.
Verfahren höherer Ordnung haben einen kleineren Approximationsfehler und sind demzufolge geeignet, genauere Ergebnisse zu erzielen.
Auf der anderen Seite muss die Funktion pro Schritt mehrfach ausgewertet werden, wodurch der Rechenaufwand wesentlich höher wird. Außerdem muss die Funktion auf dem betrachteten Intervall oft genug stetig differenzierbar sein, damit der geringere Approximationsfehler garantiert werden kann.

Heun-Verfahren

Das Heun-Verfahren ist ein Verfahren zweiter Ordnung, d.h. es besitzt einen Approximationsfehler von $O(h^2)$. Es besitzt einen Prädiktor- und einen Korrektorschritt.

Die Verfahrensvorschrift ist $x(t_{i+1}) = x(t_i) + \frac{h_i}{2} \cdot (k_1 + k_2)$
Dabei ist k_1 der Korrektor-Schritt mit $k_1 := f(t_i, x(t_i))$
Und k_2 ist der Prädiktor-Schritt mit $k_2 := f(t_{i+1}, x(t_i) + h_i \cdot k_1)$.
Der Prädiktor-Schritt bezieht also den Korrektor-Schritt mit ein.
Beim Heun-Verfahren wird das Integral im Gegensatz zum Euler-Verfahren nicht mit einem Rechteck sondern einem Trapez angenähert.

Runge-Kutta-Verfahren

Das Runge-Kutta-4-Verfahren ist ein Verfahren 4. Ordnung. Es besitzt also einen Approximationsfehler von $O(h^4)$.
Dafür muss aber die Funktion auch auf dem betrachtetetn Intervall mindestens 4-mal stetig differenzierbar sein.
Das Verfahren ist gegeben durch:

$$x(t_{i+1}) = x(t_i) + \frac{h}{6} \cdot (k_1 + 2k_2 + 2k_3 + k_4)$$

Mit den zwischenschritten:

$$k_1 = f(t_i, x(t_i))$$
$$k_2 = f(t_i + \frac{h}{2}, x(t_i) + \frac{h}{2} \cdot k_1$$
$$k_3 = f(t_i \frac{h}{2}, x(t_i) + \frac{h}{2} \cdot k_2)$$
$$k_4 = f(t_{i+1}, x(t_i) + h \cdot k_3)$$

Globaler Diskretisierungsfehler und Konvergenz

Die Abweichung der exakten Lösung und der des ESV ergibt den globalen Diskretisierungsfehler $g(t_i, x_i) := x(t_i) - x_*(t_i)$ wobei $x(t_i)$ die Näherungslösung zum Zeitpunkt t_i ist und $x_*(t_i)$ die exakte Lösung. Man spricht davon, dass das ESV die Konvergenzordnung P besitzt, wenn Pdie größte natürliche Zahl ist, für die gilt:

$$g(t_i, x_i) = x(t_k) - x_k = O(h^p)$$

Die Konvergenzordnung gibt an, wie gut die Näherungslösung also die exakte Lösung approximiert. Damit ist der globale Diskretisierungsfehler ausschlaggebend für das Konvergenzverhalten des ESV.

Lokaler Diskretisierungsfehler und Konsistenz

Hier widme man sich der Fragestellung wie gut die *Zuwachsfunktion* $\Phi(t_i, x_i, x_{i+1}, h; f)$ die eigentliche Verfahrensfunktion approximiert. Diesbezüglich verwendet man den lokalen Diskretisierungsfehler $l(t_i, x_i)$, welcher folgende Aussage über Konsistenz gibt:

> Das ESV ist konsistent mit der Ordnung P, wenn P die größte natürliche Zahl ist,
> für die gilt: $l(t_i, x_i) = \frac{x(t_{k+1}) - x(t_k)}{h} - \Phi(t_k, x_k, h; f) = O(h^p)$

Für ein konsistentes ESV folgt also, dass der Konsistenzfehler sehr klein wird und die Zuwachsfunktion die eigentliche Verfahrensfunktion für $h \longrightarrow 0$(Konsistenzbedingung) approximiert. Hier spricht man im allgemeinem von der Konsistenzordnung.

Schrittweitensteuerung

Konstante Schrittweiten haben zwei große Nachteile:
•Sie sind ungenau, wenn sich die gesuchte Lösung in einigen Bereichen sehr stark ändert, da die Schrittweite eventuell zu groß ist und dadurch bestimmte Veränderungen der Lösung nicht beachten oder zu stark beachten.

•Sie sind ineffiezient, wenn sich die gesuchte Lösung in einem Bereich nur sehr wenig ändert, da man in diesen Bereichen große Schrittweiten nutzen kann, ohne viel Genauigkeit zu verlieren.

Deswegen benutzt man oftmals eine Schrittweitensteuerung.
Aufgabe einer Schrittweitensteuerung ist es, die Schrittweite dynamisch anzupassen, so dass sie immer klein genug ist, um die gewünschte Genauigkeit zu garantieren, aber auch immer so groß, dass man keine unnötigen Schritte machen muss.
Dazu wird die Schrittweite an den lokalen Approximationsfehler angepasst, so dass dieser unter der gewollten Fhlerschranke bleibt.
Im folgenden sollen zwei übliche Ansätze für eine Schrittweitensteuerung erklärt werden.
Beim ersten Ansatz werden jeweils zwei Näherungen für x_{i+1} berechnet. Einmal mit der aktuellen Schrittweite h_i und dann nochmal mit der Schrittweite $h_i/2$. Dabei müssen mit der halbierten Schrittweite natürlich auch zwei Integrationsschritte gemacht werden, damit man beim gleichen Zeitpunkt die Ergebnisse vergleichen kann.
Aus den beiden approximierten Lösungen kann man den lokalen Fehler schätzen.
Liegt der Fehler unterhalb der vorgegebenen Toleranz, so kann der Schritt akzeptiert werden. In diesem Fall kann man mit der mit $h_i/2$ berechneten Lösung für x_{i+1} weiterrechnen. Man nimmt diese Lösung, da sie genauer ist als die mit Schrittweite h_i.
Liegt der Fehler über der vorgegebenen Toleranz, so muss man den Integrationsschritt mit einer neuen Schrittweiter wiederholen.
Der Aufwand ist bei diesem Verfahren recht hoch, da man für s Schritte des Verfahrens (gemeint sind hier die einzelnen Zwischenschritte eines Integrationsschrittes, also beim Heun Verfahren k_1 und k_2) f $s + 2s$-mal auswerten muss.
Bei der zweiten Variante betrachtet man zwei Verfahren unterschiedlicher Ordnung. Also z.B. ein Verfahren der Ordnung p und eines der Ordnung p+1.
Berechnet man nun mit beiden Verfahren mit der gleichen Schrittweite eine Näherung für x_{i+1}, so kann man aus dem Unterschied dieser Lösungen und dem Unterschied der beiden Ordnungen den lokalen Fehler schätzen.
Auch hier gilt, dass wenn der Fehler untehlab der Toleranz liegt man mit der gefunden Lösung weiterrechnet und ansonsten den Schritt mit einer neuen Schrittweite wiederholt.
Hier wird die Lösung genutzt, die mit dem Verfahren höherer ORdnung gefunden wurde, da diese genauer ist.
Bei geschickter Wahl der beiden genutzten Verfahren erhöht sich der Rechenaufwand nur sehr gering. Wählt man die Verfahren so, dass sich nur die Koeffizienten in den späteren bzw. zusätzlichen Schritten des Verfahrens höherer Ordnung unterscheiden, so muss man hat man für die ersten Schritte des zweiten Verfahrens bereits die Lösungen berchnet.
Asl Beispiel seien hier die Runge-Kutta-Verfahren 4. und 5. Ordnung genannt.
Das RK-5 Verfahren ist identisch mit dem RK4- Verfahren, bis auf einen zusätzlichen Schritt .

Hier kann man also k_1 bis k_4 vom RK-4 Verfahren nutzen und muss nur zusätzlich noch ein k_5 berechnen. Man muss also nur eine zusätzliche Funktionsauswertung pro Integrationschritt in kauf nehmen.

Der erste Ansatz hat einen wesentlich höheren Rechenaufwand. Dafür ist er aber robuster bei geringerer Ordnung.

Unstetigkeiten

Damit ein numerisches Verfahren der Ordnung p genutzt werden kann, muss die Funktion mindestens p mal stetig differenzierbar sein.
Bei Funktionen mit Unstetigkeiten ist im Bereich der Unstetigkeit diese Voraussetzung nicht gegeben. Dieses Problem lässt sich lösen, falls die Funktion auf den abschnitten zwischen den Unstetigkeiten p-mal stetig differnezierbar ist und die Stellen, an denen die Funktion unstetig ist als Ereignisse bbeschrieben werden können, die vom Zustand oder der Zeit abhängig sind.

Ursachen

Ursachen von Unstetigkeiten sind z.b.:
•Stoßvorgänge: sprunghafte Änderungen der Geschwindigkeit. Fällt z.B. ein Ball auf den Boden, so geht seine Geschwindigkeit zuerst nach unten, ab dem Moment des Aufpralls aber sofort nach oben (Kompression der Einfachheit halber ignoriert).

•Reibung in mechanischen Systemen: Übergänge zwischen Gleit- und Haftreibung, also der Reibung, wenn ein Körper über eine Oberfläche rutscht und der Reibung, die einen Körper fest an seiner Position hält. Diese Übergänge sind ebenfalls unstetig. Man stelle sich einen Würfel auf einer kippbaren Platte vor. Die Geschwindigkeit in alle Richtungen ist 0. Wird die Platte nun gekippt, so fängt der Würfel ab einer bestimmten Schräglage der platte sofort an zu rutschen, die Geschwindigkeit ändert sich also nicht stetig.

•Strukturvariable Systeme: Die Dimension des Zustands x ändert sich.

Schaltfunktionen

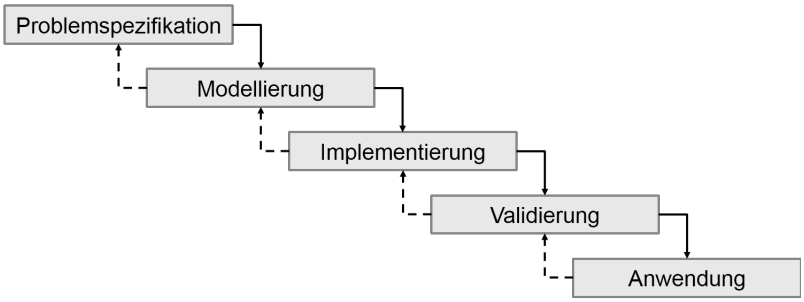
Hysterese: Ein Teil der Funktion f hängt von der Vorgeschichte von x ab, d.h. je nach der Vorgeschichte kann f verschiedene Formen annehmen. Man benutzt zeitabhängige Eingangsfunktionen und Stellgrößen um diesen Umstand zu beschreiben.$q = y$
Um den Wechsel zwischen den Zuständen der Funktion zu detektiern benutzt man Schaltfunktionen. Diese beschreiben die Stellen, an denen Unstetigkeiten auftreten.
Die Schaltfunktion hängt vom Zustand des Systems ab und muss bei den Schaltzeitpunkten eine Nullstelle haben. Angenommen es wird ein geworfener Ball betrachtet. Der Zustand sei dirch die Variablen x und y beschrieben, wobei y = h die Höhe des Bodens bescheibt und es soll geschaltet werden, wenn der Ball den Boden berührt. Die Schaltfunktion wäre damit
$q = y$

Die Variable x ist irrelevant für die Schaltfunktion, kann also ignoriert werden.
Außerdem ist die Schaltfunktion nicht eindeutig. So wäre $q = -y$ ebenfalls eine gültige Schaltfunktion, da sie die gleichen Nullstellen aufweist.
Ändert sich während eines Intergrationsschrittes das Vorzeichen der Schaltfunktion, so lieht zwischen den beiden Zeitpunkten ein Schaltzeitpunkt.
Das bedeutet, dieser muss ermittelt werden und dann muss ab diesem Zeitpunkt ein neuer Integrationsschritt gemacht werden.
Dabei muss die Schrittweite kllein genug sein um einen Vorzeichen wechsel zu detektieren. Wechselt die Schaltfunktion zwischen zwei Schritten zwei mal das Vorzeichen, so kann die Schaltfunktion dies nicht wiedergeben.

Einführung in Computational Engineering – Vorlesung 10

Autorin: Rebecca Schieren

Teilschritte einer Simulationsstudie



Problemspezifikation

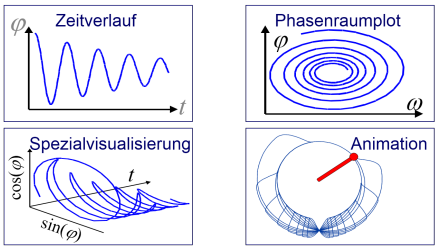
Im ersten Schritt der Simulationsstudie, der Problemspezifikation, wird als erstes die Aufgabe der Simulation formuliert und deren Ziele festgelegt. Des Weiteren werden für die Simulation Kriterien festgelegt und Daten erhoben.

Modellierung

Die Modellierung ist der zweite Schritt in der Simulationsstudie. Hier wird die Realität in einem Modell abgebildet. Dazu wird die Struktur des Modells festgelegt, Modellgleichungen aufgestellt und das Modell anschließend noch vereinfacht. Der Ausgangspunkt der Modellbildung ist die Festlegung der Zustandsvariablen. Die Systemstruktur besteht aus den Eingangsgrößen, dem Systemzustand, den Ausgangsgrößen und Parametern.

Implementierung

Im dritten Schritt der Simulationsstudie wird für das jeweilige Modell ein Berechnungsverfahren ausgewählt bzw. entwickelt. Anschließend werden Modell und Berechnungsverfahren programmiert und die Berechnungsergebnisse visualisiert. Als Visualisierung bieten sich Zeitverlauf, Phasenraumplot, Spezialvisualisierungen und Animationen an. Falls Notwendig erfolgt noch eine Laufzeitoptimierung des Programms.



Validierung

Bei der Validierung wird überprüft, ob die Simulationsergebnisse auf die Realität übertragbar sind. Siehe dazu Vorlesung 1. Des Weiteren dient sie der Fehlersuche, der Konsistenzüberprüfung und dem Daten- und Parameterabgleich. Die Validierung ist notwendig, da wärend der Simulationsstudie verschiedene Fehler auftreten können. Unter anderem Modellierungsfehler, Approximationsfehler der iterativen Berechnungsverfahren, Rundungsfehler und Implementierungsfehler.

Anwendung

Die Simulationsergebnisse können dazu genutzt werden, das reale Verfahren anzupassen durch Variation der Parameter und der Struktur und zur Optimierung bzw. Vorhersage.

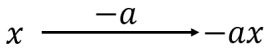
Blockorientierte Darstellung

Die Blockorientierte Darstellung kann zur Implementierung in Programmen wie z.B. Matlab bzw. Simulink genutzt werden. Es ist eine Darstellung für Differntialgleichungen n-ter Ordnung als Graph, bei dem die einzelnen Knoten die Komponenten der Rechenvorschriften der Differentialgleichung sind.
Die Komponenten haben in der Regel einen oder mehrere Eingänge und einen Ausgang. Im folgenden werden die verschiedenen Komponenten erklärt:

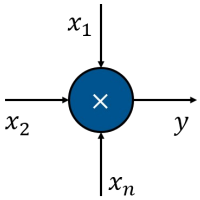
•Zustandsvariablen:

•Parameter oder Eingaben:

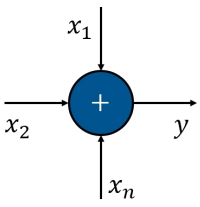
•Faktoren:

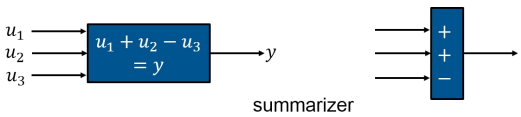


•Produkt über alle Eingangsvariablen:



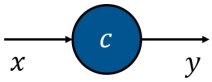
$$y = \prod x_i$$

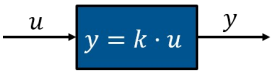


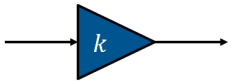


•Summe über alle Eingangsvariablen - rechts in der Simulinkdarstellung:

$$y = \sum x_i$$





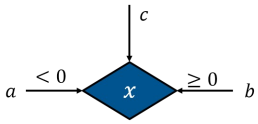


gain

•Gain (Multiplikation mit einer Konstanten) - rechts in der Simulinkdarstellung:

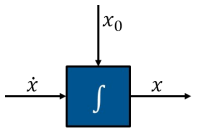
$$y = c \cdot x$$

•Schaltfunktionen:

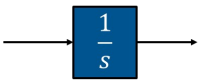


if $c < 0$ then $x = a$
if $c \geq 0$ then $x = b$

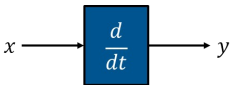
•Integration (rechts in der Simulinkdarstellung):



integrator



•Differentiation:



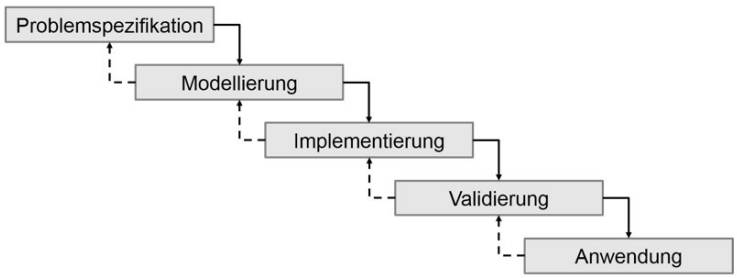
$y = \frac{dx}{dt}$

Einführung in Computational Engineering – Vorlesung 11

Autoren: Jan Stengler und Chinara Mammadova

Interpretation und Validierung

Validierung



Die Validierung ist ein wichtiger Teilschritt in einer Simulationsstudie. In diesem Schritt wird getestet, ob die Simulationsergebnisse auf die Realität übertragbar sind. Weiterhin werden systematische Plausibilitätsüberprüfungen, Fehlersuchen, Konsistenzprüfungen sowie Daten- und Parameterabgleiche durchgeführt. Das zu validierende Simulationsmodell entspricht der Implementierung von Modell und Berechnungsverfahren.
Die Validierung ist die Begründung der These, dass ein Simulationsmodell innerhalb seines Anwendungsbereiches einen ausreichenden Genauigkeitsbereich hat, welcher konsistent ist mit den anvisierten Anwendungen des Simulationsmodells (Sargent 2003).
Weiterhin ist sie der Prozess der Bestimmung zu welchem Grad ein Simulationsmodell eine genaue Repräsentation der realen Welt ist, betrachtet aus der Perspektive der beabsichtigten Anwendungen des Simulationsmodells (Office of the Director of the U.S. Defense Research and Engineering, 2002).
Hierbei wird eine Reihe von Tests sukzessive durchgeführt. Wenn diese erfolgreich waren, wird parallel dazu die Vertrauenswürdigkeit des Simulationsmodells stetig erhöht. Dies liegt daran, dass keine einzelnen Tests existieren, mit denen die Validität eines Simulationsmodells nachgewiesen werden kann.

Verifikation

Die Verifikation ist ein formaler Nachweis der Korrektheit, dass ein Programm einer vorgegebenen Spezifikation entspricht. In den meisten Fällen ist es nicht möglich die vollständige Korrektheit eines kontinuierlich dynamischen Simulationsmodells zu beweisen. Dies liegt an der unendlich großen Anzahl von Zustandsverläufen und Störungseinflüssen nichtlinearer dynamischer Systeme. Mit Hilfe einiger systematisch und erfolgreich untersuchen Beispielen und Tests kann die Wahrscheinlichkeit der Korrektheit erhöht werden. Diesem Ziel dient die Validierung.

Plausibilitätsüberprüfung

Die Plausibilitätsüberprüfung ist eine Überprüfung, dass ein Programm einer vorgegebenen Spezifikation entspricht. Der Nachweis der ausrechenden Glaubwürdigkeit des Simulationsmodells im Hinblick auf dessen Einsatzbereich ist ein weiteres Ziel der Validierung.

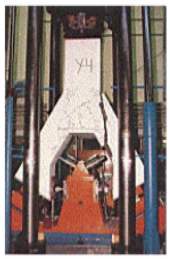
Warum ist die Validierung wichtig?

Durch die Validierung wird die Gefahr von fehlerhaften Aussagen der Simulationsstudien stark reduziert. Diese fehlerhaften Aussagen können Fehlentscheidungen mit möglicherweise verheerenden Folgen verursachen. Im folgenden wird ein Beispiel für solch einen Fehler mit schwerwiegenden Folgen gezeigt.

Beispiel: Ölbohrplatform "Sleipner A" (1991)

Die Struktur der Ölbohrplatform "Sleipner A" wurde innerhalb von drei Jahren (1988-1991) entwickelt. Hierbei wurde ein "State of the Art" Simulationsprogramm verwendet, das bereits bei Vorgänger-Plattformen eingesetzt wurde.
Am 23.8.1991 wurde die 110 Meter hohe Beton-Tiefseestruktur durch Einpumpen von Wasser abgesenkt. Während dieses Vorgangs ertönte ein rumpelndes Geräusch, woraufhin die Plattform unkontrolliert ins 220 Meter tiefe Wasser absank und dabei komplett zerstört wurde. Der Wert der zerstörten Beton-Tiefseestruktur hat 180 Millionen US$ betragen. Der entstandene Gesamtschaden belief sich auf ca. 700 Millionen US$.





Die Betonstruktur bestand aus 24 Zellen, von denen 4 zu Säulen verlängert wurden. Die inneren Wände wurden durch 32 dreieckförmige Trizellen verbunden. Die Trizellen sind mit einem gängigen Finite-Elemente Simulationsprogramm simuliert und konstruiert worden. Dabei war die Berechnung der Schwerkraft ungenau (um 47% zu gering), was bei früheren Plattformen unkritisch war. Jedoch wurden dadurch die Trizellen zu dünn ausgelegt, weshalb die Plattform eingestürzt ist.
Verbesserte Simulationen konnten in der Analyse des Unglücks ein Leck in 62 Metern Tiefe vorhersagen. Die reale Tiefe, in der das Leck aufgetreten ist, hat 65 Meter betragen.
Weitere "klassische" Beispiele für Fehler in der Simulationsstudie sind der "Eichtest" der Mercedes-Benz A-Klasse (1997) und die Londoner Millenium Brücke (2000).

Fehlerquellen bei Modellierung & Simulation

Während der Modellierung und der Simulation können folgende Fehler entstehen:
•Modellierungsfehler: Fehler durch vereinfachende Modellannahmen (z.B. starrer statt elastischer Körper) und Ungenauigkeiten in Modellparametern

•Approximationsfehler des iterativen Berechnungsverfahren (z.B beim Euler-Verfahren proportional zur Schrittweite)

•Rundungsfehler: Fehler durch Ausführung des Berechnungsverfahrens auf Computer mit endlicher Zahldarstellung

•Programmier-, Implemmentierungsfehler

In den Teilschritten einer Simulation wird in der Problemspezifikation ein Problem P spezifiziert. In der Modellierung entsteht der Modellierungsfehler |P - M|. Während der Implementierung treten der Diskretisierungsfehler |M-D|, der Abbruch-&Rundgsfehler |D-L| und der Visualisierungsfehler |L-V| auf. Dadurch kann der Gesamtfehler |P-V| = |P-M| + |M-D| + |D-L| + |L-V| berechnet werden. Eine Lösung L wird dann akzeptiert, wenn in allen 4 Schritten vergleichbar kleine Fehler gemacht wurden.
Da "Simulationsmodell = Implementierung von Modell + Berechnungsverfahren" gilt, müssen in der Validierung die Implementierung, das Modell und das Berechnungsverfahren untersucht werden, um festzustellen, dass das Simulationsmodell das reale System im Rahmen der Zielsetzung ausreichend gut abbildet.

Validierung der Implementierung

Die syntaktische Fehlerfreiheit der Implementierung kann z.B. mit Hilfe eines Debuggers getestet werden. Weiterhin sollten Plausibilitätsüberprüfung der Simulationsergebnisse für Spezialfälle („naturgetreues“ Verhalten) durchgeführt werden. Ein Beispiel hierfür ist eine von der Ruhelage ausgelenkte, leere Schiffschaukel. Diese pendelt eine Weile hin und her (Gravitationseinfluss) bevor sie zur Ruhe (Reibungseinfluss) kommt. Die numerische Korrektheit der Implementierung kann z.B. durch den Vergleich der mit dem Simulationsmodell berechneten Lösung und einer analytischen Lösung für einen Spezialfall (z.B. bei der Schiffschaukel durch ein einfaches Pendel mit gegebenen Daten) überprüft werden.

Validierung des Modells

In der Validierung des Modells sollte die Zulässigkeit und logische Konsistenz der Modellannahmen (und deren Anwendbarkeit zur Problemlösung) überprüft werden. Zum Beispiel kann das in der Vorlesung untersuchte Starrkörpermodell der Schiffschaukel möglicherweise die Bewegungsdynamik sehr zuverlässig abbilden, aber ungeeignet sein, wenn andere Aspekte untersucht werden sollen, wie z.B. Strukturbelastungen und Materialermüdungen. Es sollte die ausreichende Detailliertheit des Modells (und korrekte Modellstruktur) getestet werden. Bei der Schiffschaukel z.B. das starrer, nicht veränderliches Drehzentrum / starre gegen etwas elastisches Gestänge / radiale Bewegungen einer einzelnen Person gegen radiale und tangentiale Bewegungen mehrerer Personen / ...). Ebenso sollte die Korrektheit der Modellparameter geprüft werden. Bei der Schiffschaukel z.B. die Masse, die Gravitationskonstante, der Abstand der Schaukel vom Drehzentrum und die Reibungskonstante.

Validierung des Berechnungsverfahrens

Bei der Validierung des Berechnungsverfahrens sollte überprüft werden, ob die numerische Lösung für das Modell geeignet ist. Zum Beispiel sind bei Modellen mit steifen Differentialgleichungen nur implizite und nicht explizite Integrationsverfahren geeignet. Der Approximationsfehler des (iterativen) Berechnungsverfahrens sollte geprüft werden. Zum Beispiel sollten Integrationsverfahren höherer Ordnung (z.B. Runge-Kutta 7./8. Ordnung) bei hohen Genauigkeitsanforderungen verwendet werden und Verfahren niedriger Ordnung (z.B. Euler-Verfahren) bei niedrigen Anforderungen. Weiterhin muss getestet werden, ob eine adaptive Schrittweitensteuerung oder konstante Schrittweite verwendet werden sollte. Ebenso muss der Einfluss von Rundungsfehlern untersucht werden, z.B. sollte kritische Programmteile auf Auslöschung untersucht werden (Subtraktion oder Division etwa gleich großer Zahlen).

Tests auf Plausibilität und Konsistenz

Im folgenden werden einige Tests auf Plausibilität und Konsistenz gezeigt.
•Reproduktion von beobachtetem bzw. „natürlichem“ Systemverhalten: Wie gut stimmt das mit Hilfe des Simulationsmodell berechnete Verhalten mit (in der Vergangenheit) beobachtetem bzw. bekannten Erkenntnissen über das Systemverhalten überein?

•Vorhersage von Verhalten: Wie plausibel ist das simulierte Systemverhalten für ein fiktives / noch nicht beobachtetes Szenario und können mit dem Simulationsmodell die Ergebnisse eines geplanten realen Experiments in ausreichender Genauigkeit vorhergesagt werden?

•Verhaltensanomalien: Ein stark unterschiedliches bzw. widersprüchliches Verhalten des Simulationsmodells gegenüber dem realen System weist auf signifikante Fehler in Modell oder Berechnungsverfahren oder Implementierung hin.

•Systemverhalten in Extremsituationen: Entspricht das Verhalten des Simulationsmodells auch bei Simulation extremer Szenarien, die beim realen System selten oder gar nicht auftreten, den Erwartungen/Erfahrungen?

•Parametervariationen und Parametersensitivität: Ist das Verhalten des Simulationsmodells sensitiv zu plausiblen Variationen in den Werten der Modellparameter?

Zusammenfassend gilt, dass die Validierung fachliche Einsicht und Kreativität benötigt. Es gibt keine Pauschallösungen für die Validierung.

Einführung in Computational Engineering – Vorlesung 12

Autoren: Chinara Mammadova und Jan Stengler

Systemidentifikation

Das Ziel der Systemidentifikation ist die Bestimmung des Parameter θ des Modells vom System $\dot{x} = f(x, \theta)$

Die Hauptschritte der Systemidentifikation sind Strukturidentifikation/Validierung und Parameteridentifikation. Das Ziel der Validierung ist zu überprüfen, ob die Funktion f mit gesammelten Daten übereinstimmt, d.h. ob $\dot{x} = f(x, \theta)$ ist. Das Ziel der Parameteridentifikation besteht darin, bei der vorgegebenen Struktur den Parameter θ zu bestimmen. Die Systemidentifikation besteht aus den folgenden fünf Schritten.

Schritt 1: Spezifikation

In diesem Schritt wird die Modellklasse spezifiziert und parametrisiert.
$\dot{x} = f(x, \theta)$, wobei $\dot{x}$ die Ausgabe, x die Eingabe und θ der Parameter ist.

Schritt 2: Datenerhebung

In diesem Schritt werden Datenpaare bestehend aus Eingaben und Ausgaben gesammelt.
$D = \{(\dot{x}_i, \hat{x}_i) | i \in 1, 2, ..., m\}$

Zudem werden Daten zufällig in Trainingsdatensatz $D_{train} = randomSelection(D)$ und Testdatensatz $D_{test} = D \setminus D_{train}$ unterteilt.

Schritt 3: Parameteridentifikation

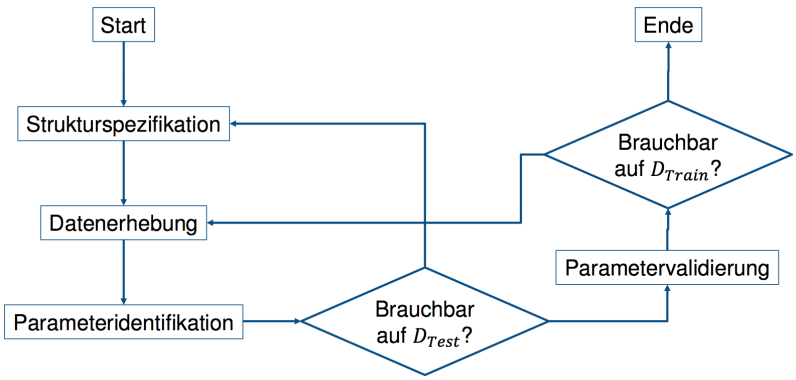
In diesem Schritt werden die optimalen Parameter bestimmt.
$\theta^* = \underset{\theta}{argmin}\{Kosten(D_{train}, \theta)\}$

Schritt 4: Strukturvalidierung

Dieser Schritt überprüft, ob Kosten(D_{train}, θ)>Kostenspezifikation. Wenn dies der Fall ist, wird zum Schritt 1 zurückgegangen.

Schritt 5: Parametervalidierung

Dieser Schritt bestimmt, ob Kosten(D_{test}, θ)≤Kostenspezifikation. Wenn dies nicht der Fall ist, wird zum Schritt 3 zurückgegangen. Ansonsten werden die Parameter übernommen.



Im Folgenden wird die Parameteridentifikation ausführlicher erläutert. Die Grundidee hierbei ist, den Fehler in den Messwerten zu minimieren. Angenommen
$$y = \Phi\theta + \epsilon$$

Dabei ist
$$y = \begin{bmatrix} \hat{x_1} \\ \hat{x_2} \\ \vdots \\ \hat{x_m} \end{bmatrix}$$
Φ ist eine (m x n)-Matrix von Basisfunktionen (Features).
$$\Phi = \begin{bmatrix} \phi_1(\hat{x_1}) & \dots & \phi_n(\hat{x_1}) \\ \phi_1(\hat{x_2}) & \dots & \phi_n(\hat{x_2}) \\ \vdots & \ddots & \vdots \\ \phi_1(\hat{x_m}) & \dots & \phi_n(\hat{x_m}) \end{bmatrix}$$
θ ist ein n-dimensionaler Vektor aus Parameters.
$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix}$$
ϵ ist ein m-dimensionaler Vektor von Rauschen.
$$\epsilon = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_m \end{bmatrix}$$

Der quadratische Fehler berechnet sich folgendermaßen:
$$Kosten(D, \theta) = \frac{1}{2}\sum_{i=1}^{n} \|e_i\|^2 = \frac{1}{2}\sum_{i=1}^{n} e_i^T e_i = \frac{1}{2}(y - \Phi\theta)^T(y - \Phi\theta)$$

Wird dieser minimiert, so ist
$$\theta = (\Phi^T\Phi)^{-1}\Phi^T y$$