Antworten (SySi)

1 Unterschiedliche Interpolationsverfahren

1.1 Da sowohl der Anfangswert (0) als auch der Endwert (10) mitgezählt werden, hat *u* 11 Elemente.

1.2 Aus den gleichen Gründen hat *uu* 101 Elemente.

1.3 *yl* hat die gleiche Anzahl von Elementen wie der feiner unterteilte Stützstellenvektor *uu*, nämlich 101.

1.4 size (*yl*) liefert *1 101*, also einen Zeilenvektor.

1.5 In diesem Beispiel kommt der Spline dem Originalsinus am nächsten. Das Polynom dritten Grades ist etwas schlechter als der Spline aber wesentlich besser als die lineare Interpolation.

1.6 Splines sind nicht unbedingt immer die ideale Interpolationsfunktion. Ein Polynom dritten Grades beispielsweise läßt sich natürlich am besten durch ein Polynom dritten Grades interpolieren.

1.7 Genauso kann eine lineare Interpolation optimal sein, wenn bekannt ist, daß sich die Originalfunktion aus Geradenstücken zusammensetzt (Rechteck, Dreieck, Sägezahn, ...)

1.8 Die im Intervall $[0,10]$ geplottete Kurve hat bei $x = \pi$ einen Ordinatenwert von ungefähr 30.

1.9 Für π selbst ist die Funktion singulär ($-\infty$). Etwa ab dem Intervall $[3, 3.2]$ zeigt sich eine kleine negative Zacke, die auf eine Unstetigkeit hindeutet.

2 Zweidimensionale Interpolation

2.1 Die 7 x 7 Abtastung mit bilinearer Interpolation läßt die Originalfunktion nur noch erahnen.

2.2 Die kubische Interpolation kommt dem Original schon ziemlich nahe. Für die aus e-Funktionen und Polynomen bestehende Peak-Funktion stellt sie natürlich eine weitaus bessere Näherung dar als die lineare Interpolation.

2.3 Wie der Name schon nahelegt, erhält jeder Interpolationspunkt den Wert seines nächsten (Stützstellen-)Nachbarn. Es entstehen also quadratische Plateaus (gleicher Amplitude) um die Stützstellen herum.

3 Regression (Best Fit)

3.1 Die *randn*-Funktion liefert bei jedem Aufruf wieder einen ganz neuen Zufallswert. Wenn dies nicht gewünscht wird, kann der Zufallsgenerator mit *randn ('state', 0)* vorher auf einen definierten Wert initialisiert werden.

3.2 Je weniger Punkte, desto größer die Chance, daß ein paar zufällige Ausreißer die Regressionsgerade weit weg von der Originalgerade plazieren.

4 Schwinger zweiter Ordnung

4.1 *k* ist der stationäre Verstärkungsfaktor der Übertragungsfunktion, also der Wert für „t gegen unendlich“, bzw. „s gegen null“. *gain* resultiert aus folgender Übertragungsfunktionsdarstellung:

$$F(s) = \text{gain}\frac{(s - z_1)(s - z_2)\cdots(s - z_n)}{(s - n_1)(s - n_2)\cdots(s - n_m)}$$

Der Wert *k* für „s gegen null“ ist dann

$$\lim_{s\to 0}(F(s)) = \text{gain}\frac{(-z_1)(-z_2)\cdots(-z_n)}{(-n_1)(-n_2)\cdots(-n_m)}$$

4.2 Wie der Name schon sagt, stellt *pzmap* nur die Pole und die Nullstellen dar. Ein Verstärkungsfaktor ändert aber nichts an der Lage von Polen und Nullstellen.

4.3 In der Zustandsraumdarstellung der Vorlesung wird die Eingangsgröße direkt mit dem Faktor *k*wo*wo* multipliziert und der zweite Zustand (Ausgang des rechten Integrators) ist identisch mit der Ausgangsgröße. MATLAB hingegen gibt die Eingangsgröße direkt auf den Eingang des ersten Integrators und multipliziert dafür den zweiten Zustand mit dem Faktor. Die numerischen Werte der Zustände unterscheiden sich daher; das Eingangs-Ausgangs-Verhalten ist aber in beiden Fällen identisch. Allgemein gibt es beliebig viele Zustandsraumdarstellungen, die sich durch Zustandstransformationen ineinander umrechnen lassen und die alle die gleiche Übertragungsfunktion haben.

4.4 Die beiden gezeichneten Sprungantworten (Übertragungsfunktion und Zustandsraumdarstellung) gehören zu demselben System, sind daher identisch und liegen übereinander.

4.5 Die Gerade ist die von MATLAB freundlicherweise berechnete Asymptote der Sprungantwort für „t gegen unendlich“.

4.6 Bei der in *step* verwendeten „Integrationsmethode“ (Transitionsmatrix) ist der berechnete Wert zu den Abtastzeitpunkten immer identisch, egal, mit welcher Schrittweite „integriert“ wurde.

4.7 Je mehr Punkte ermittelt werden, desto länger dauert verständlicherweise die Berechnung, aber desto detaillierter wird natürlich auch die Kurve.

4.8 Eigenwerte, Frequenzen und Dämpfungen sind Systemparameter, die einzig in der System- oder Dynamikmatrix *amat* enthalten sind. Sie beschreiben nur das Eigenverhalten des Systems, also die Reaktion auf Anfangszustände. Die restlichen Matrizen (*bmat*, *cmat* und *dmat*) hingegen definieren die Reaktion des Systems auf Eingangsgrößen.

5 Identifikation im Zeitbereich

Keine Fragen, keine Antworten ...

6 P-T2 Simulation

6.1 *Scope* zeichnet schon während der Simulation. Wenn etwas schiefläuft, kann dies sofort erkannt und die Simulation abgebrochen werden. *Scope* stellt alle Kurven in vorgeschriebenen Farben als durchgezogene Linien mit komfortabler Zoom-Funktionalität dar; Einzelpunkte sind nicht möglich. *plot* ist eine Offline-Darstellung, kann also erst nach Beendigung der gesamten Simulation angestartet werden. Weitergehende Diagrammbearbeitungen wie Achsenbeschriftungen, Kopieren in die Zwischenablage, ... sind nur *plot* in möglich.

6.2 Das Signal springt von einem Abtastzeitpunkt zum nächsten beispielsweise von null auf eins. Da aber der „null"-Abtastzeitpunkt auf der Abszisse eine Sekunde weiter links liegt als der „eins"-Abtastzeitpunkt, ergibt sich die geneigte Verbindungsgerade (Polygonzug). Die Flankensteilheit wird desto größer, je kleiner das Schrittweite ist. Auch die Sprungantwort wird nur jede Sekunde berechnet.

6.3 Je kleiner die Schrittweite ist, desto genauer wird das Ergebnis und desto länger dauert die Simulation und desto länger werden die Datengräber. Bei extrem kleinen Schrittweiten schlägt der Rundungsfehler der Maschine zu, was in der Praxis aber nur bei steifen Systemen erreicht wird. Abhilfe: Nur dort mit kleiner Schrittweite integrieren, wo auch tatsächlich etwas passiert -> Variable Schrittweite

6.4 Es ist deutlich zu erkennen, wie die Schrittweite zwischen zwei Flanken langsam und kontinuierlich bis zum Defaultwert von $\frac{t_{stop} - t_{start}}{50} = \frac{20 - 0}{50} = 0.4$ zunimmt, um den geforderten Maximalfehler nicht zu überschreiten.

7 Gruppieren und Maskieren

7.1 Sie selbst, als Sie die Maskierungs-Dialogbox ausgefüllt haben.

7.2 Auch den Help-Text haben Sie selbst eingegeben und können ihn jederzeit nach Aufruf von *Edit Edit Mask* im Feld *Documentation Block help* modifizieren.

7.3 *<p>* ist HTML-Syntax und bewirkt einen Zeilenumbruch.

7.4 Die *Drawing commands* zeichnen die Block-Ikone; in diesem Fall die Sprungantwort des jeweiligen P-T_2.

7.5 Wenn Sie den Dämpfungsparameter des P-T_2 ändern, wird die Sprungantwort auf der Block-Ikone automatisch angepaßt.

7.6 Eine nachträgliche Änderung in einem Block der Bibliothek wirkt sich automatisch auf alle „Klone“ des Bibliotheksblockes aus.

8 Mathematisches Pendel

8.1 Sie haben vergessen, die Integratoren auf einen von null verschiedenen Wert zu initialisieren. Ansonsten bleibt das Pendel während der gesamten Simulation in seiner Ruhelage senkrecht nach unten hängen.

8.2 Sie lenken das Pendel aus seiner Ruhelage aus, halten es bei einem bestimmten Anfangswinkel fest und lassen los.

8.3 Nur eine recht geringe Differenz zwischen beiden Kurven sichtbar.

8.4 Für die meisten Anwendungsfälle ist die lineare Näherung daher ausreichend.

8.5 Bei einem größeren Anfangswinkel wird der Linearitätsbereich verlassen, die lineare Näherung weicht schon nach wenigen Perioden signifikant vom nichtlinearen Original ab und kann allenfalls noch für qualitative Betrachtungen herhalten.

8.6 Die lineare Näherung ist nur in unmittelbarer Nähe des „Arbeitspunktes“, um den herum linearisiert wurde, gültig. Je weiter man sich vom Linearisierungspunkt entfernt, desto schlechter stimmt das lineare System mit dem nichtlinearen Original überein.

8.7 Die magentafarbene Kurve, also die langsamere, die mit der kleineren Frequenz stammt vom nichtlinearen Pendel. Anschauliche Erklärung: Die Rückführung des Winkels auf den Eingang des Integrators der Winkelgeschwindigkeit bestimmt im Wesentlichen die Frequenz des Schwingers. Je größer diese Rückkopplungsverstärkung, desto schneller schwingt das System. Da aber der Sinus, der beim nichtlinearen Pendel in der Rückführung steht, betragsmäßig immer kleiner als die „eins“ des linearen Pendels bleibt, scheint es plausibel, daß das nichtlineare System langsamer schwingt.

8.8 Mit einem Winkel von *pi* bildet das nichtlineare System ein „aufrecht stehendes Pendel“ in seinem labilen Gleichgewichtszustand. Wenn nun aber keine Störungen auftreten und auch der Integrationsalgorithmus keine numerischen Schmutzeffekte beiträgt, kann das Pendel theoretisch beliebig lange in seiner labilen Ruhelage verharren. Das linearisierte Pendel kennt solch eine Ruhelage überhaupt nicht. Es beschreibt einfach einen ungedämpften linearen Schwinger zweiter Ordnung, der unabhängig von der Anfangsamplitude mit einem reinen Sinus reagiert.

8.9 Die geringe Anfangsstörung braucht geraume Zeit, um das Pendel aus seiner labilen Ruhelage herauszubewegen. Dann schwingt es schnell durch die untere Ruhelage und erreicht (fast) wieder den oberen Totpunkt. Auch dort dauert es dann ziemlich lange, bis die kleine Störung zu einem Rückschwingen des Pendels führt. Andere Integrationsverfahren und -parameter können an dieser Stelle übrigens aufgrund numerischer Fehler auch dazu führen, daß das Pendel überschlägt. Probieren Sie's ruhig mal aus ...

8.10 Das nichtlineare Pendel schwingt, da es ja schon mit 90 Grad über dem oberen Totpunkt hinaus gestartet wird, natürlich weiter in positive Richtung, so daß eine Schwingung mit einer Amplitude von 90 Grad um 360 Grad herum zustande kommt. Im jetzt völlig falschen linearen Modell existiert diese Periodizität des Winkels überhaupt nicht. „Folgerichtig“ schwingt das lineare Pendel mit einer Amplitude von 270 Grad um seine einzige Ruhelage bei null Grad.

8.11 Das Pendel rauscht zum Anfangszeitpunkt gerade mit einer endlichen Geschwindigkeit durch die untere Ruhelage.

8.12 Die Gesamtenergie aus potentieller (oberer Totpunkt) und kinetischer Energie (Anfangsgeschwindigkeit) reicht aus, um das nichtlineare Pendel immer wieder überschwingen zu lassen. Der Winkel steigt und steigt und ... Das lineare Pendel produziert seinen üblichen Sinus.

8.13 Klar!

8.14 Ein lineares Modell hat nur in der Nähe des Linearisierungspunktes Gültigkeit. Weiter weg kann der Fehler *beliebig* groß werden. Hier: Solange der Sinus ungefähr gleich seinem Argument ist:

$$\sin(0.00) = 0.0000$$
$$\sin(0.01) = 0.0100$$
$$\sin(0.10) = 0.0998$$
$$\sin(0.50) = 0.4797$$
$$\sin(1.00) = 0.8415$$
$$\sin(10.0) = -0.5440$$

9 Trimmrechnung

9.1 Der Algorithmus trimmt einen Winkel von 0.7854 rad aus, was genau den geforderten 45 Grad entspricht. Der Sinus davon wird mit negativem Vorzeichen zurückgeführt und kompensiert dann am Eingang des Winkelgeschwindigkeitsintegrators gerade den Eingangstrimmwert von „eins durch Wurzel zwo“. Da die Winkelgeschwindigkeit selbst auch zu null getrimmt wird, sind alle Integratoreingänge null und das System bleibt statisch.

9.2 In der verwendeten Syntax trimmt der Algorithmus alle x-Punkte (Ableitungen, Integratoreingänge, linke Seiten der Differentialgleichungen) zu null. Der Trimmzustand ist statisch. Nichts tut sich. Mit einer erweiterten Trimmsyntax ist es aber auch möglich, stationäre Arbeitspunkte, bei denen nicht alle Ableitungen verschwinden, auszutrimmen (Beispiel: Flugzeug im unbeschleunigten, horizontalen Kurvenflug, Drehgeschwindigkeit ungleich null, Drehwinkel nimmt linear zu).

9.3 Die ausgetrimmten Zustände werden zwar als richtige Anfangswerte an den Integrationsalgorithmus übergeben; leider fehlt aber der Anfangswert der Eingangsgröße.

9.4 So geht‘s.

9.5 Ja.

9.6 Siehe Anwort 8.9.

10 Linearisierung

10.1 Die Blockschaltbilder sind identisch.

10.2 Bei 90 Grad hat das linearisierte System keine Rückführung mehr.

10.3 Es entsteht eine Reihenschaltung zweier Integratoren, ein doppelter Integrator oder I_2-Glied.

10.4 Auch die Taylorreihenentwicklung führt zu einem doppelten Integrator.

11 Integrationsverfahren

11.1 Nein, die Sprungantwort sieht vernünftig aus.

11.2 Der Pol liegt bei -0.1 auf der negativen reellen Achse, also noch im Stabilitätskreis, ziemlich weit am rechten Rand.

11.3 Jetzt liegt der Pol bei -1 genau im Mittelpunkt des Kreises auf der reellen Achse. Das System wird also mit Sicherheit noch stabil simuliert.

11.4 Das Simulationsergebnis hat allerdings jetzt schon nicht mehr viel mit der üblichen e-Funktion einer P-T_1-Sprungantwort zu tun. Vielmehr ergibt sich eine Rampe von null auf eins in einer Sekunde.

11.5 Wenn das System bei gleicher Schrittweite doppelt so schnell gemacht wird (T = 0.5) rutscht der Pol aus der Mitte an den linken Rand des Stabilitätsgebiets auf einen Wert von -2.

11.6 Wie zu erwarten war, ergibt sich eine grenzstabile Schwingung, die sich hier als Dreiecksschwingung ausbildet.

11.7 Schon bei T = 0.499 ergibt sich eine leicht aufklingende Schwingung, die allerdings erst sichtbar wird, wenn man einige Perioden simuliert (z. B. 50 Sekunden lang).

11.8 Ohne Schwierigkeiten.

11.9 Obwohl der Integrator am rechten Rand des Stabilitätsgebietes liegt.

11.10 Die einzig wahre Aussage ist die, daß Pole im Inneren des Stabilitätsgebiets stabil, Pole auf dem Rand grenzstabil und Pole außerhalb instabil simuliert werden. Andererseits werden aber langsame Pole „richtiger“ simuliert als schnelle Pole. So wird beispielsweise der Integrator, der ja praktisch ein unendlich langsamer Tiefpaß ist, völlig *richtig* grenzstabil simuliert. Ein Pol im Mittelpunkt des Stabilitätsgebiets hingegen wird zwar stabil, aber ziemlich unrealistisch simuliert.

11.11 Das konjugiert komplexe Polpaar liegt außerhalb des Stabilitätsgebiets auf der imaginären Achse. Das ungedämpfte P-T_2, das eigentlich eine genzstabile Sinusschwingung als Sprungantwort haben sollte, wird also von Euler instabil simuliert.

11.12 Die aufklingende Schwingung ist natürlich um den Faktor 10 langsamer, da das Polpaar jetzt viel dichter am Ursprung liegt und auch das Aufklingen selbst geschieht moderater; aber auch diese eigentlich grenzstabile Schwingung wird instabil simuliert.

11.13 Euler kann keinen ungedämpften Schwinger stabil simulieren. Die komplette imaginäre Achse liegt außerhalb des Stabilitätsgebiets.

11.14 Leonard Euler (1707 – 1783) war ein großartiger Mathematiker; das nach ihm benannte Integrationsverfahren sollte hingegen nur in Ausnahmefällen verwendet werden, zum Beispiel, wenn man keine Ahnung oder keine Zeit hat, ein besseres Integrationsverfahren zu implementieren.

11.15 1 rad/s liegt gerade in der „Stabilitätsbeule“, mit der einige Runge-Kutta-Verfahren grenz- und sogar leicht instabile Systeme fälschlicherweise *stabil* simulieren. Um es ganz deutlich zu machen: Diese Erweiterung des Stabilitätsgebietes in den instabilen Bereich hinein ist ein *Nachteil* des Integrationsverfahrens. Er führt genauso zu *falschen* Simulationsergebnissen, wie das instabile Simulieren eigentlich stabiler Systeme anderer Verfahren (vgl. Euler)

11.16 Wenn die Pole näher an den Ursprung heranrücken (kleinere Frequenz) kann RK4 den grenzstabilen Schwinger über recht viele Perioden mit hinreichender Genauigkeit ohne Amplitudenveränderung simulieren.

11.17 Ja, abgesehen von der geringen Abtastrate von 2π-mal pro Periode, die den Kurvenverlauf recht zackig aussehen läßt, bleibt die Schwingung über sehr lange Zeit vorschriftsmäßig grenzstabil.

12 Steife Systeme

12.1 Alle Integrationsverfahren simulieren das steife System praktisch gleich gut. Der schnelle Pol ($T_1 = 1$) liegt schließlich bei allen Verfahren noch im Stabilitätsgebiet und wird daher stabil, schnell und im Simulationsergebnis nicht sichtbar simuliert.

12.2 Bei Euler liegt der schnelle Eigenwert direkt auf der Stabilitätsgrenze, was zu einer hochfrequenten Schwingung mit geringer Amplitude führt, die sich der langsamen e-Funktion des langsamen Pols überlagert. Die anderen Verfahren simulieren den schnellen Pol noch stabil.

12.3 Euler wird eindrucksvoll instabil. Die anderen Verfahren offensichtlich nicht.

12.4 Auch das Verfahren zweiter Ordnung (Heun) steigt erwartungsgemäß außerhalb seines Stabilitätsrandes von -2 aus (bei einer Schrittweite von
2.01 müssen Sie allerdings schon 2000 Sekunden simulieren, bevor die Instabilität erkennbar wird. Das Verfahren dritter Ordnung (Bogacki-Shampine) kann gerade mit einer Schrittweite von 2.5 Sekunden noch stabil simulieren und fängt bei 2.51 schon an, erste Oberschwingungen zu zeigen. RK4 explodiert irgendwo bei einem Integrationsintervall zwischen 2.7 und 2.8. Das Verfahren fünfter Ordnung (Dormand-Prince) gibt jenseits von 3.3 auf.

13 Begrenzungen

13.1 Sobald das Eingangssignal kleiner als die Begrenzung geworden ist, beginnt der Steller, dem jetzt kleineren Kommando zu folgen.

13.2 Hier gibt es eine Totzeit zwischen der Umkehr des Kommandos und der Reaktion des begrenzten Stellers.

13.3 Wenn der Steller in die Sättigung läuft, wird nur die Ausgangsgröße begrenzt, der Integrator aber läuft weiter „voll“, bis auf den gleichen Wert, den er ohne den Begrenzerblock erreicht hatte. Der am Ausgang zusätzlich angeschlossene Begrenzer hat schließlich keine Rückwirkung auf das P-T_1 selbst. Wenn nun das Kommando plötzlich kleiner als der Grenzwert wird, muß der Integrator sich erst in normaler Tiefpaßmanier (e-Funktion) entladen. Es dauert also eine geraume Zeit, bis der Ausgang des Integrators wieder kleiner als der Grenzwert geworden ist und der begrenzte Wert ebenfalls beginnt, dem Kommando zu folgen.

13.4 Im Prinzip brauchen Sie nur einen Integrator, der, wenn sein Ausgang eine vorgegebene Grenze überschreitet, *angehalten* wird, dessen Eingang dann also „gegrounded“ (zu null gesetzt) wird. Um das mit einzelnen Blöcken zu realisieren, wären ein Schalter, logische Vergleicher, ein paar Konstanten, und natürlich ein Integrator notwendig.

13.5 Klar!

13.6 Wenn die Geschwindigkeit begrenzt , also konstant ist, steigt bzw. fällt die Position als Integral einer konstanten Größe linear.

13.7 Leider ist das Gesamtsystem durch die Einführung einer *Saturation* oder eines *Rate Limiter*s *nichtlinear* geworden. Die Reihenfolge der Blöcke wird somit signifikant. Stellen Sie sich beispielsweise vor, das P-T_1 wäre so langsam, daß seine Sprungantwort nie schneller als die im *Rate Limiter* vorgegebene Rate wird. Dann ist es egal, ob hinter dem P-T_1 noch ein *Rate Limiter* sitzt, da dieser sowieso nieangesprochen wird. Das Gesamtsystem reagiert also auf einen Sprung mit der unverfälschten Sprungantwort des P-T_1. Wenn aber der *Rate Limiter* vor dem P-T_1 sitzt, wandelt er den Sprung als erstes in eine Rampe um, die dann als Eingangssignal des P-T_1 wirkt. Die Antwort des P-T_1 ist dann nicht wie im vorherigen Fall die Sprungantwort, sondern die Rampenantwort, die ja genau das Integral der Sprungantwort darstellt.

13.8 Nur der *Saturation*-Block am Eingang des Integrators begrenzt wirklich genau die Geschwindigkeit des Stellers, da ja nur der Integratoreingang als Differential der Position am Ausgang tatsächlich exakt der Geschwindigkeit entspricht.

13.9 Auf diese Weise lassen sich ratenbegrenzte Systeme leider nicht einfach aus Standardblöcken (*Rate Limiter* + *Transfer Fcn*, ...) zusammensetzen, sondern müssen umständlich aus *Integrator*s, *Gain*s, *Sum*s und *Saturation*s zusammengebastelt werden. Andererseits kann man sich natürlich auch eine kleine Bibliothek mit positions-, geschwindigkeits- und beschleunigungsbegrenzten P-T_1 und P-T_2 aufbauen (vgl. Gruppieren und Maskieren).