

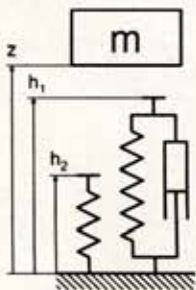
# Root-Funktionen

## Die Integration nicht stetig differenzierbarer Zustandsverläufe

SIMPACK bietet dem Anwender zahlreiche außerordentlich leistungsfähige Integratoren zur Berechnung von Zustandsverläufen. Root-Funktionen sorgen für minimale Rechenzeit und optimale Rechengenauigkeit bei Knicken und Sprüngen in der Lösung.

Gerhard Hippmann, INTEC GmbH

Intelligente Integratoren wie ODASSL/SODASRT oder LSO-DE arbeiten mit variabler Schrittweite und Interpolationsordnung. Zu deren Steuerung kann stets nur auf bekannte Ergebnisse ("Vorgeschichte") zurückgegriffen werden, wobei von stetig differenzierbaren Lösungen ausgegangen wird. Treten im System jedoch Ereignisse auf, die einen Knick im Verlauf der Zustandsgrößen  $x(t)$  bewirken, muß der Integrator unter relativ hohem Rechenaufwand Schrittweite und Ordnung umstellen. Außerdem ergeben sich an der Unstetigkeitsstelle Schwierigkeiten bei der Einhaltung der gewünschten Fehlerschranken. Um dennoch maximale Performance und optimale Rechen-



nente RINFO(i) reserviert (z.B. kefel (7,nrf) = 2).

### Initial Call

Beim Start der Integration findet einmalig der "Initial Call" statt. Die Werte RINFO (i) werden gleich Null gesetzt und die Kraftelemente im "Initial State"  $x(t_{init})$  aufgerufen. Für RINFO = 0 liefern Kraftelemente mit Roots ihren Output auf herkömmliche Weise, indem z.B. die Kraftwirkung aus Messungen zwischen den Koppelmarkern, einer Fallunterscheidung (Kontakt ja/nein) und dem zugehörigen Kraftgesetz berechnet wird. Darüber hinaus wird das Ergebnis der Bereichszuordnung als Werte 1 oder 2 in RINFO abgelegt. Im Beispiel könnte RINFO wie in der Tabelle gesetzt werden.

Im Initial State  $x(t_{init})$  darf sich das System nicht genau an einer Unstetigkeitsstelle befinden, da sonst keine Zuordnung möglich bzw. RINFO undefiniert ist. Bei der Programmierung von User Routinen ist zu beachten, daß in ufel (User Defines Force Element) nur beim Initial Call Änderungen an RINFO vorgenommen werden.

### Integration

Während der Integration hängt die Wahl des Kraftgesetzes ausschließlich vom RINFO-Vektor ab. Im Beispiel könnte in ufel stehen:

```
IF ( RINFO(1) .EQ. 1 )
THEN
    F = 0.0
ELSE IF ( RINFO(2)
.EQ. 1 )
```

```
F = c1*(z-h1) +
d1*zp
ELSE
    F = c1*(z-h1) +
d1*zp + c2*(z-h2)
END IF
```

Die eigentliche Schaltfunktionalität wird anhand der Root-Finding-Funktionen  $s_{x(t)}$  verwirklicht. Jedes Integrationsergebnis wird auf Vorzeichenwechsel dieser algebraischen, stetigen Funktionen mit genau einem Nulldurchgang überprüft. Im Beispiel könnte die erste Root-Finding-Funktion  $s_{1(x(t))} = z - h1$  lauten. Durchläuft der Körper bei der Integration die Lage  $z = h1$ , stellt der Integrator einen Vorzeichenwechsel bei  $s_1$  fest. Die Integration wird angehalten und mittels  $s_{1(x(t))}$  anhand eines schnell konvergierenden Iterationsverfahrens der Zeitpunkt der Unstetigkeit  $t_{root}$  berechnet; das Resultat liegt stets geringfügig nach  $t_{root}$  vor. Bevor die Integration mit kleiner Schrittweite und Ordnung 1 wieder ab  $t_{root}$  startet, wird RINFO (1) z.B. von 1 nach 2 umgeschaltet, so daß mit veränderter rechter Seite weitergerechnet wird. Außerdem besteht an dieser Stelle die Möglichkeit, Zustandsgrößen gezielt zu verändern (näheres im SIMPACK Training).

Die Verwendung von Root-Funktionen wird durch die Integratoren ODASRT/SODASRT und LSO-DAR unterstützt. Schaltet der Benutzer die Root-Funktionalität aus oder verwendet er einen anderen Integrator, werden die Kraftelemente stets mit RINFO = -1 betrieben und die Kraftgesetze

Bereich	Lage z	RINFO(1)	RINFO(2)
I	$z > h1$	1	1
II	$h2 < z < h1$	2	1
III	$z < h2$	2	2

genauigkeit zu bieten, stellt SIMPACK Root-Funktionen zur Verfügung.

Nicht stetig differenzierbare Lösungen  $x(t)$  resultieren aus Unstetigkeiten in der "rechten Seite" des zu integrierenden DGL-Systems. Ein typischer Anwendungsfall (Kontaktproblem) ist im Bild dargestellt:

Je nach Lage des in vertikaler Richtung beweglichen Körpers gelten drei verschiedene Kraftgesetze. Die Bereiche I, II und III (siehe Tabelle) werden anhand des RINFO-Vektors unterschieden. Im Pre-Processing wird für jede Unstetigkeit eine Kompo-

wie auch beim Initial Call auf herkömmliche Weise angewendet. Die Measurements (Work Values AFEL) werden ebenfalls mit  $RINFO=-1$  berechnet und dürfen daher nicht von RINFO abhängen.

Weitere Hinweise

- "Zuschalten" eines Dämpfers durch Kontakt ergibt einen Sprung im Kraftverlauf und folglich einen Knick im Geschwindigkeitsverlauf (siehe Beispiel).

- Bei Anwendung der Stoßgesetze entspricht der Kraftverlauf einem Dirac-Impuls, was einen Sprung im Geschwindigkeits- und einen Knick im Lageverlauf bewirkt.
- Sollen Systeme mit unstetiger rechter Seite ohne Root-Funktionen integriert werden, eignen sich Integratoren mit Schrittweiten- und/oder Ordnungssteuerung kaum; relativ kleine Fehler sind dann mit nied-

riger Ordnung und ggf. konstanter Schrittweite zu erwarten.

- Die User-Routine `ufel/ufun40` ist ein gutes Beispiel zur Programmierung von Kraftelementen mit Root-Funktionen.

Für Details gehen wir gerne während des SIMPACK User-Trainings auf Root-Funktionen ein.

## Tuning des Symbolic Code in SIMPACK 7.0

Trotz beeindruckendem Preis/Leistungs-Verhältnis der jüngsten Generation von Microprozessoren ist die Minimierung der Rechenzeiten von Simulationsprogrammen nicht überflüssig, denn mit der Abnahme der Rechenzeit steigt die Gestaltungsfreiheit des Ingenieurs in der Erstellung der Modelle. Zusammen mit der Parametervariation ergeben sich neue Perspektiven bei der Auslegung eines Systems. Mit SIMPACK 7.0 wurde das Modul *Symbolic Code* von Grund auf neugestaltet.

Claus Schwientek, INTEC GmbH

### Das Symbolic Code - Prinzip

SIMPACK kann die Bewegungsdifferentialgleichungen eines Systems anstelle der numerischen in symbolischer Form generieren. Die entstehenden Gleichungen werden auf die Zahl der zur Berechnung benötigten Operationen hin optimiert.

SIMPACK verwendet für Berechnungen im Allgemeinen den numerischen Code, der für eine große Klasse von mechanischen Modellen die Bewegungsgleichungen in allgemeiner Form enthält. Im numerischen Code werden die zur Zeitschrittintegration benötigten Abläufe durch "IF-Abfragen" und "DO-Loops" gesteuert.

Der symbolische Code hingegen wird für ein spezielles Modell optimiert. Er enthält keine unnötigen Berechnungen oder "IF-Abfragen", sondern nur noch die für ein bestimmtes Modell nötige Minimalzahl an Operationen. Modell und Integrationsverfah-

ren sind zu einem ausführbaren Programm kondensiert.

Die Verwendung des modellspezifisch optimierten Codes erfordert weder beim Aufbau eines Modells, beim *Post-Processing*, noch bei der Konfiguration des Integrators ein Umdenken, sondern beschränkt sich auf das Klicken eines anderen Buttons. Der Aufwand für die Erstellung des *Symbolic Code* wurde mit SIMPACK 7.0 deutlich reduziert, so daß sein Einsatz nunmehr bei wesentlich kleineren Modellen rentabel wird.

### Anwendungen

Der modellspezifisch optimierte Code umfaßt Modelle mit starren und elastischen Körpern. Bei der Erstellung der symbolischen Gleichungen wird für die meisten Gelenk-Typen ein spezieller symbolischer Code erzeugt, für die übrigen Gelenke und die Kraftelemente erfolgt die Berechnung über Aufrufe von Unterprogrammen, den SIMPACK

"Library-" oder "USER-" Routinen. Somit ist die Möglichkeit gegeben, beim symbolischen Code genau wie beim numerischen Code selbst programmierte User-Routinen zu verwenden.

### Gewinne

Den größten Leistungsgewinn bringt der symbolische Code, wenn im Modell hauptsächlich einachsige Gelenke verwendet werden. Rechenzeitgewinne bis Faktor 30 wurden bereits erzielt. Im Hinblick auf den SIMPACK-Mehrkörperformalismus, bei dem der Rechenaufwand linear mit der Anzahl an Körpern steigt, ist dieser Code somit für **Realtime-Simulationen** bestens geeignet. Die Zahl der Varianten eines Systems, die mit der Parametervariation innerhalb eines gegebenen Zeitraums ausgeleuchtet werden können, wirkt sich zu einem echten Wettbewerbsvorteil aus, genau wie die Zeitersparnis für die modellseitige Rechenzeitoptimierung.