stageverslag

REDUCTIEMETHODEN VOOR MULTI-BODYSYSTEMEN

Rob Heythuysen

Rapport WFW 93.166

datum: 21 oktober 1993
id. no.: 301807
begeleider: Ted van de Broek
docent: Fons Sauren
Vakgroep: WFW
Sectie: Dynamica
Hoogleraar: D.v.Campen

INHOUDSOPGAVE

	pag.
LITERATUURLIJST	1
INLEIDING	3
HOOFDSTUK 1	
1.1 Inleiding	4
1.2 Bewegingsvergelijkingen van een Multibody-systeem met constraints	4
HOOFDSTUK 2	
Partitionering van gegeneraliseerde coördinaten	
2.1 Inleiding	7
2.2 Toepassing GCP met LU-decompositie op gekoppeld stelsel DAE's	7
2.3 Voor- en nadelen van de GCP-methode	11
2.4 Conclusies	11
HOOFDSTUK 3	
De Constraint Violation Stabilization methode	
3.1 Inleiding	12
3.2 Toepassing CVS-methode op gekoppeld stelsel DAE's	13
3.3 Evaluatie van de CVS-methode	14
HOOFDSTUK 4	
Een Hybride numerieke integratie methode	
4.1 Inleiding	15
4.2 Het hybride numerieke integratie algoritme	15
4.3 Evalutie van de hybride integratie methode	16
HOOFDSTUK 5	
De Singuliere Waarden Decompositie	
5.1 Inleiding	17
5.2 Basiseigenschappen van de SWD	18
5.3 Partitionering van gegeneraliseerde coördinaten m.b.v. de SWD	19
5.4 Evaluatie van de SWD	22
5.5 Conclusies	22
HOOFDSTUK 6	
De Projectie Methode	
6.1 Inleiding	23
6.2 Introductie van de projectie methode	24
6.3 De Projectiemethode voor systemen met holnome constraints	26
6.3.1 Inleiding	26
6.3.2 Het Gram-Schmidt proces en de NS	28
6.3.3 Berekening van de onafhankelijke coördinaten en de gemodificeerde versnellingsvergelijking	29
6.4 Algoritme voor de projectiemethode	32
6.5 Toelichting bij het algoritme	37
6.5.1 Uitleg van het programma	37
6.6 Conclusies	39

pag.

APPENDIX A

A1 De projectiemethode; inleiding 40
A2 Probleemformulering 40
A3 De wiskundinge projectiemethode formulering 43
A4 De projectiemethode geassocieërd met holonome constraints 46
A5 De projectiemethode geassocieërd met niet-holonome constraints 48

APPENDIX AA

AA1 Kromlijnige coördinaten 50
AA2 Covariant en Contravariant 51
AA3 Uitwerking van formule (4.4) 54

APPENDIX B

Coördinatenpartitionering met LU-decompositie 55

APPENDIX C

Uitbreiding van de afleiding in §6.3.3 57

APPENDIX D

Listing van het programma:"numeriek algoritme voor de projectiemethode" (31 pag.), geschreven in Fortran(F77 59

BIJLAGE

Gebruikte subroutines uit de Fortran NAG-bibliotheek behorende bij het algoritme voor de projectiemethode

LITERATUURLIJST

[1] R.A.Wehage en E.J.Haug, **Gegeneralized coordinate partitioning for dimension reduction in analysis of constraidynamic systems**, ASME Jnl. of mech. design (104), pag. 247, 1982

[2] E.J.Haug, **Computer aided kinematics and dynamics of mechanical systems**, pag. 243, 1989

[3] T.W.Park en E.J.Haug, **A Hybrid numerical integration method for machine dynamic simulation**, ASME Jnl. of mech. transm. and automation in design (108), pag. 211, 1986

[4] P.E.Nikravesh en I.S.Chung, **Application of Euler parameters to the dynamic analysis of three dimensional constrained dynamic analysis**, ASME Jnl. of mech. design (104), pag. 785, 1982

[5] Gilbert Strang, **Linear algebra and its applications**, 1986

[6] C.L.Singh en P.W.Likins, **Singular Value Decomposition for constrained dynamic systems**, ASME Jnl. of applied mechanics (52), pag. 943, 1985

[7] Stageverslag Edwin Starmans, intern rapport, no. WFW 87.010

[8] N.K.Mani, E.J.Haug en K.E.Atkinson, **Application of singular value decomposition for analysis of mechanical system dynamics**, ASME Jnl. of mech. trans. and automation in design (107), pag. 82, 1985

[9] N.K.Mani en E.J.Haug, **Singular value decomposition for dynamic system design sensitivity analysis**, Engineering with computers (1), pag. 103, 1985

[10] Dan Scott, **Can a projection method of obtaining equations of motion compete with Langrange's equations?**, American Jnl. Physics (56), no. 5, pag. 451, 1988

[11] Joel Storch en Stephen Gates, **Motivating Kane's method for obtaining equations of motion for dynamic systems**, Jnl. Guidance (12), no.4, pag. 593, 1989

[12] W.Blajer, **A projection method approach to constrained dynamic analysis**, ASME Jnl. of applied mechanics (59), pag. 643, 1992

[13] C.G.Liang en G.H.Lance, **A differentiable null space method for constrained dynamic analysis**, ASME Jnl. of mech. transm. and automation in design (109), pag 405, 1987

[14] E.Haug en Tchung, **Methods of Engineering Mathematics**, the University of Iowa, 1993

[15] Shampine en Gorden, **Computer solution of ordinary differential equations, "The initial value problem**

[16] James W. Longley, **Least squares computations using orthogonalisation methods**

INLEIDING

De beschrijving van de bewegingsvergelijkingen in de multibody dynamica leidt tot een stelsel gekoppelde algebraïsche (de constraintvgl.) en differentiaalvgl (de bewegingsvgl.). Impliciete numerieke integratie methoden voor het oplossen van een gemengd systeem van algebraïsche en differentiaalvergelijkingen (DAE's) leiden tot numerieke moeilijkheden geassocieërd met slecht geconditioneerde matrices [1].

Voor regeltechnische toepassingen is de beschrijvingswijze m.b.v. DAE's niet erg handig. Liever zouden we een stelsel onafhankelijke differentiaalvergelijkingen hebben die het systeemgedrag beschrijven.

In de literatuur worden diverse methoden beschreven om het stelsel gekoppelde DAE's te reduceren maar een stelsel onafhankelijke vergelijkingen.

Eigenlijk zijn er 2 principiële benaderingen om de bewegingsvergelijkingen van zulke systemen op te lossen [6].

(a) Men kan onbekende krachten of momenten introduceren op de interacties tussen constrained lichamen (vaak symbolisch aangegeven met Lagrange multipliers), en dan de dynamische vergelijkingen gelijktijdig met de constraintvergelijkingen oplossen om de constrained krachten en momenten als ook de kinematische variabelen te bepalen.
(b) Als alternatief, kan men de constraintvergelijkingen gebruiken om de dimensie van het systeem van de dynamische vergelijkingen die opgelost moeten worden te reduceren, en daarbij de behoefte elimineren van het representeren of oplossen van constrained krachten en momenten.

In de volgende hoofdstukken zullen een aantal methoden, zoals ze beschreven zijn in de literatuur en betrekking hebben op bovenstaande 2 benaderingen, geïntroduceerd en uitgewerkt worden zodat men een stelsel onafhankelijke differentiaalvergelijkingen overhoudt die m.b.v. numerieke integratietechnieken gemakkelijker oplosbaar zijn.

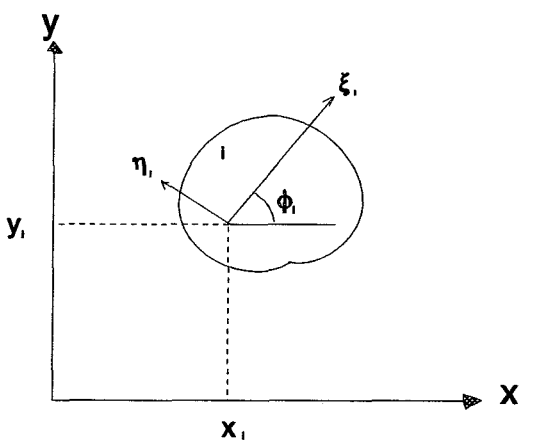
HOOFDSTUK 1

1.1 Inleiding.

In de volgende hoofdstukken zullen een aantal methoden geanalyseerd en besproken worden die een gekoppeld stelsel van DAE's reduceren tot een stelsel van onafhankelijke differentiaalvgl., en zodoende t.g.v. de numerieke integratiemethoden resultaten te verkrijgen die een minimale numerieke fout bevatten. Alvorens in te gaan op een aantal methoden zal eerst het gekoppeld stelsel van DAE's afgeleid worden.

1.2 Bewegingsvgl. van een Multibody-systeem met constraints.

Een typisch star lichaam i in een vlak is te zien in figuur 1.



figuur 1. Coördinatensysteem in 2D

Het massamiddelpunt van dit lichaam wordt beschreven t.o.v. een inertieël X - Y carthesisch coördinaten stelsel door zijn x en y coordinaten resp. x_i en y_i. Een lichaamsgebonden ξ_i-η_i coördinaten stelsel, met z'n oorsprong in het massamiddelpunt, wordt gebruikt om elk punt te localiseren in het lichaam i. De oriëntatie van het lichaam m.b.t. de initiële assen wordt gespecificeerd door de hoek ϕ_i.
Dan is de vector met geg. coördinaten van lichaam i

$$q_i = [x_i \;\; y_i \;\; \phi_i]^T \tag{1.2.1}$$

en de corresponderende vector met gegeneraliseerde krachten

$$Q_i = [Q_x^i \;\; Q_y^i \;\; Q_\phi^i]^T \tag{1.2.2}$$

Hierin zijn Q_x^i en Q_y^i de X en Y componenten van de resulterende kracht die werkt op het massamiddelpunt van het lichaam i en is Q_ϕ^i het resulterende moment dat werkt op het lichaam.

De complete vector van gegeneraliseerde coördinaten van het systeem wordt geschreven als

$$q_{(n*1)} = [q^{1^T}\ q^{2^T}\ .\ .\ q^{NB^T}]^T \tag{1.2.3}$$

De bijbehorende vector met gegeneraliseerde krachten wordt geschreven als

$$Q_{(n*1)} = [Q^{1^T}\ Q^{2^T}\ .\ .\ Q^{NB^T}]^T \tag{1.2.4}$$

waarin NB = het aantal lichamen en $n = 3NB$.

De kinematische constraintvgl. tussen de lichamen in het systeem kunnen geschreven worden als

$$\Phi(q, t) = 0 \tag{1.2.5}$$

met Φ een m-vector van constraint functies, die uitziet als

$$\Phi = [\Phi_1(t, q), \ .\ .\ .\ , \Phi_m(t, q)]^T \tag{1.2.6}$$

(1.2.5) 1x differentiëren levert de snelheidsvergelijking:

$$\dot{\Phi}(q, t) = \Phi_{,q}\dot{q} + \Phi_{,t} = 0 \Leftrightarrow \Phi_{,q}\dot{q} = -\Phi_{,t} = v \tag{1.2.7}$$

Onderschriften betekenen partiële afgeleiden t.o.v. de afgebeelde variabele. Hier betekent het dus:

$$\Phi_{,q} = \left[\frac{\partial \Phi_i}{\partial q_j}\right]_{mxn} \quad ; \quad \Phi_{,t} = \left[\frac{\partial \Phi_i}{\partial t}\right] \tag{1.2.8}$$

Nogmaals (1.2.5) differentiëren levert de versnellingsvergelijking:

$$\begin{aligned} \ddot{\Phi}(q, t) &= \Phi_{,q}\ddot{q} + (\Phi_{,q}\dot{q})_{,q}\dot{q} + 2\Phi_{,qt}\dot{q} + \Phi_{,tt} = 0 \\ \Rightarrow \Phi_{,q}\ddot{q} &= -(\Phi_{,q}\dot{q})_{,q}\dot{q} - 2\Phi_{,qt}\dot{q} - \Phi_{,tt} = \gamma \end{aligned} \tag{1.2.9}$$

Voor het stelsel van gekoppelde algebraïsche differentiaalvgl. van een constrained dynamisch systeem volgt nu:

$$\begin{bmatrix} M & \Phi_{,q}^{T} \\ \Phi_{,q} & 0 \end{bmatrix} \begin{bmatrix} \ddot{q} \\ \lambda \end{bmatrix} = \begin{bmatrix} Q^{A} \\ \gamma \end{bmatrix} \tag{1.2.10}$$

Hierin is M de nxn massamatrix van het systeem, $\Phi_{,q}$ is de mxn constraint Jacobi-matrix, λ is de m-vector met de corresponderende Lagrange multiplicatoren, en Q^{A} is een n-vector met de gegeneraliseerde krachten.

Behalve dat de positievgl. (1.2.5) en snelheidsvgl. (1.2.7) moeten gelden op het begintijdstip t_0, moeten er ook extra beginvoorwaarden voor de positie en snelheid geintroduceerd worden.

Voor resp. de positie en de snelheid vindt men dan:

$$q(t_0) = q_0 \quad \Rightarrow \quad \Phi(q_0, t_0) = 0 \tag{1.2.11}$$

$$\dot{q}(t_0) = \dot{q}_0 \quad \Rightarrow \quad \Phi_{,q}(q_0, t_0)\,\dot{q}_0 = -\Phi_{,t}(q_0, t_0) = -v_0 \tag{1.2.12}$$

Vgl. (1.2.11) en (1.2.12) houden in dat q_0 en $\dot{q}_0$ consistent moeten zijn met de constraints. Dus q_0 moet voldoen aan vgl. (1.2.5) en $\dot{q}_0$ moet voldoen aan vgl. (1.2.7).

Het is dus belangrijk dat de beginvoorwaarde voor de positie uit (1.2.11) en (1.2.5), berekend op t_0, eenduidig de beginpositie $q(t_0)$ bepaalt. Eveneens is het belangrijk dat de beginvoorwaarde voor de snelheid uit (1.2.12) en (1.2.7), berekend op t_0, eenduidig de beginsnelheid $\dot{q}(t_0)$ bepaalt.

In de volgende hoofstukken zullen een aantal methodes behandelt worden die de gekoppelde DAE's (1.2.10) reduceren naar een stelsel onafhankelijke vergelijkingen.

De volgende methoden zullen achtereenvolgens behandelt worden:

- Partitionering van Gegeneraliseerde Coördinaten (G.C.P)
- De Constraint Violation Stabilization methode (C.V.S)
- Een Hybrid algoritme: een kruising van G.C.P. en C.V.S.
- De Singuliere Waarden Decompositie (S.W.D.)
- De Projectie-methode.

HOOFDSTUK 2

Partitionering van Gegeneraliseerde Coördinaten (GCP)

2.1 Inleiding.

Hier wordt een computer-gebaseerde methode voor de formulering en het efficiënt oplossen van niet-lineaire, constrained differentiële bewegingsvgl. voor mechanische systemen beschreven. Niet-lineaire holonome constraintvgl. en differentiële bewegingsvgl. worden geschreven in termen van een maximale set van carthesische gegeneraliseerde coördinaten, om de algemene formulering van constraint- en krachtfuncties te vereenvoudigen.
De GCP leidt tot een eenvoudige constraint formulering in termen van carthesische coördinaten, en tenslotte resteert een minimaal systeem van differentiële bewegingsvgl. die gelijk in aantal zijn aan het aantal graden van vrijheid van het systeem.
Het algoritme bestaat uit Gaus-eliminatie met full pivoting, gevolgd door LU-decomposite op de constrained Jacobimatrix $\Phi_{,q}$,

- om het aantal vrijheidsgraden van het systeem te bepalen
- de partitionering van de geg. coördinaten in een set van afhankelijke en onafhankelijke coördinaten te bepalen
- voor constructie van een influence coëfficiënten matrix die de afhankelijke snelheden in termen van de onafhankelijke snelheden uitdrukt. (**Appendix B**)

Deze informatie wordt aangewend om numeriek een gereduceert stelsel van differentiële bewegingsvgl. te construeren, wiens oplossing de totale dynamische responsie behelst.

2.2 Toepassing GCP met LU-decomp. op gekoppeld stelsel DAE's.

Het stelsel gekoppelde DAE's wordt in een ietwat andere vorm geschreven t.o.v. het afgeleide stelsel in hoofdstuk 1.

$$\begin{cases} M(q)\ddot{q} + \Phi_{,q}^{T}\lambda = Q^{A}(q,\dot{q},t) \\ \Phi(q,t) = 0 \end{cases} \tag{2.2.1}$$

$$\Phi_{,q}\dot{q} = v \tag{2.2.2}$$

$$\Phi_{,q}\ddot{q} = \gamma \tag{2.2.3}$$

Uit bovenstaande volgt dat de constraintvgl. moeten voldoen aan vgl. (2.2.2) en de snelheidsvgl. moeten voldoen aan vgl. (2.2.3), met begincondities zoals in vgl. (1.2.11) en vgl. (1.2.12). Vgl. (2.2.1) vormt een systeem met $3n$ 2^{e}-orde differtiaalvgl. en m algebraïsche vgl.

Als men dit stelsel van gekoppelde DAE's wil oplossen met bijv. directe integratie is dit zeer gecompliceerd (oplossen van een grote set niet-lineaire vergelijkingen) en vergt dit een grote berekeningstijd. Hier zal dit gekoppeld stelsel DAE's naar een minimale dimensie worden gereduceerd, zodat het stelsel gemakkelijker oplosbaar wordt.

De matrix $\Phi_{,q}(q,t)$ heeft volle rang als de constraints onafhankelijk zijn, dus er is altijd op z'n minst één reguliere submatrix van $\Phi_{,q}$ met rang m. Gaus-Jordan reductie van de martrix $\Phi_{,q}$ met dubbel pivoting (**Appendix B**) definieërt een partitionering van $q=(u^T,v^T)^T$ zodat $\Phi_{,u}$ de submatrix van $\Phi_{,q}$ is, wiens kolommen corresponderen met elementen u van q en $\Phi_{,v}$ de submatrix van $\Phi_{,q}$ is, wiens kolommen corresponderen met elementen v van q.

----------Intermezzo----------

Bekijk een virtuele verplaatsing δq die voldoet aan de constraintvgl. $\Phi_{,q}\delta q=0$. Deze vgl. kan m.b.v. Gaus-Jordan reductie getransformeerd worden naar

$$U\delta u + R\,\delta v = 0$$

voor een waarde q die voldoet aan de 2e vgl. van (2.2.1).
Omdat de coëfficiëntenmatrix U een driehoeksmatrix is met eenheidswaarden op de diagonaal (U is regulier), kan δu dus eenduidig bepaald worden voor gekozen waarden δv.
Vanwege dit feit kan $\boldsymbol{v}$ worden geinterpreteerd als een vector met **onafhankelijke gegeneraliseerde coördinaten** en $\boldsymbol{u}$ is een vector met **afhankelijke gegeneraliseerde coördinaten.**
U is een matrix die verkregen wordt door elementaire rij operaties op $\Phi_{,u}$, dus $\Phi_{,u}$ is regulier.
Hierdoor kan $\Phi(q,t)$ theoretisch opgelost worden voor u als functie van v en t, dus $u=u(v,t)$.

----------Einde intermezzo----------

(2.2.2) kan geschreven worden in gepartioneerde vorm als

$$\begin{aligned} &\boldsymbol{\Phi}_{,q}\dot{q}+\boldsymbol{\Phi}_{,t}=0 \\ \Leftrightarrow\ &\boldsymbol{\Phi}_{,u}\dot{u}+\boldsymbol{\Phi}_{,v}\dot{v}=-\boldsymbol{\Phi}_{,t} \end{aligned} \tag{2.2.4}$$

en omdat $\Phi_{,u}$ regulier is geldt

$$\begin{aligned} \dot{u} &= -\boldsymbol{\Phi}_{,u}^{-1}\boldsymbol{\Phi}_{,v}\dot{v}-\boldsymbol{\Phi}_{,u}^{-1}\boldsymbol{\Phi}_{,t} \\ &= D\dot{v}-\boldsymbol{\Phi}_{,u}^{-1}\boldsymbol{\Phi}_{,t} \quad ;D=-\boldsymbol{\Phi}_{,u}^{-1}\boldsymbol{\Phi}_{,v} \end{aligned} \tag{2.2.5}$$

De 1e vgl. van (2.2.1) kan nu op dezelfde manier gepartioneerd worden zoals met de geg. coördinaten q gebeurd is. Er volgt:

$$\begin{cases} M^{uu}\ddot{u}+M^{uv}\ddot{v}=Q^{Au}-\Phi_{,u}{}^{T}\lambda \\ M^{vu}\ddot{u}+M^{vv}\ddot{v}=Q^{Av}-\Phi_{,v}{}^{T}\lambda \end{cases} \tag{2.2.6}$$

Omdat $\Phi_{,u}$ regulier is, mag men de 1e vgl. van (2.2.6) gebruiken om λ te elimineren uit de 2e vgl. van (2.2.6). Er resteert dan vgl. (2.2.7) in de vorm

$$(M^{vv}+D^{T}M^{uv})\,\ddot{v}\;+(M^{vu}+D^{T}M^{uu})\,\ddot{u}=Q^{Av}+D^{T}Q^{Au} \tag{2.2.7}$$

Nu kan (2.2.3) op dezelfde manier gepartioneerd worden zoals hierboven. Er resteert dan

$$\Phi_{,u}\ddot{u}+\Phi_{,v}\ddot{v}=\gamma=-(\Phi_{,q}\dot{q})_{,q}\dot{q}-2\Phi_{,qt}\dot{q}-\Phi_{,tt}$$

$$\Rightarrow\quad \ddot{u}=D\ddot{v}-\Phi_{,u}{}^{-1}[(\Phi_{,q}\dot{q})_{,q}\dot{q}+2\Phi_{,qt}\dot{q}+\Phi_{,tt}] \tag{2.2.8}$$

(2.2.7) kan nu expliciet geschreven worden als een 2e-orde differtiaalvgl. in termen van de onafhankelijke gegeneraliseerde coördinaten v door (2.2.8) in te vullen in (2.2.7):

$$\begin{aligned}(M^{vu}+D^{T}M^{uu})\,(D\ddot{v}-\Phi_{,u}{}^{-1}[(\Phi_{,q}\dot{q})_{,q}\dot{q}+2\Phi_{,qt}\dot{q}+\Phi_{,tt}])\;+\\ +(M^{vv}+D^{T}M^{uv})\,\ddot{v}\;=\;Q^{Av}+D^{T}Q^{Au}\end{aligned} \tag{2.2.9}$$

Om deze bewegingsvgl. te integreren is het handiger om ze te schrijven in een 1e-orde vorm. Dit wordt gedaan door een vector s van onafhankelijke snelheden te definiëren als

$$\dot{v}\;=\;s \tag{2.2.10}$$

en de totale vector $\dot{q}$ van gegeneraliseerde snelheden te schrijven in termen van s, m.b.v. (2.2.5) als volgt

$$w(s)\;=\dot{q}(s)\;=\begin{bmatrix}\dot{u}\\ \dot{v}\end{bmatrix}=\begin{bmatrix}Ds-\Phi_{,u}{}^{-1}\Phi_{,t}\\ s\end{bmatrix} \tag{2.2.11}$$

De resulterende bewegingsvgl. kunnen nu beschreven worden als een combinatie van (2.2.9) en (2.2.11) als

$$[(M^{vv}+D^TM^{uv}) + (M^{vu}+D^TM^{uu})D]\,\dot{s} = Q^{Av}+D^TQ^{Au}+ \\ + (M^{vu}+D^TM^{uu})\,\Phi_{,u}{}^T[(\Phi_{,q}W)_{,q}W+2\Phi_{,qt}W+\Phi_{,tt}] \qquad (2.2.12)$$

(2.2.10) en (2.2.12)) vormen nu een set van $6n-2m$ 1e-orde differentiaalvgl. in de onafhankelijke gegeneraliseerde coordinaten v en snelheden s. Alle termen in (2.2.1) worden bepaald als functie van v en s door de constraintvgl. $\Phi(q,t)=0$ en (2.2.11). Oplossen van dit systeem geeft de complete systeemtoestand, incl. Lagrange multipliers als de constrained krachten gewenst zijn. Door integratie van (2.2.10) en (2.2.12) m.b.v de beginvoorwaarden

$$q(0) = q^0 \\ \dot{q}(0) = \dot{q}^0 \qquad (2.2.13)$$

orden v en s bepaald. Uit de constrainedvgl. $\Phi(q,t)=0$ wordt u bepaald, (2.2.5) bepaalt $\dot{u}$, en $\ddot{u}$ kan berekend worden uit (2.2.8). Tenslotte kan λ bepaald worden uit de 1e vgl. van (2.2.6).

NB. De matrixinverse Φ_u^{-1} wordt nooit uitgerekend. De matrix $\Phi_{,q}$ wordt omgezet met rij- en kolompivoting in de matrices L^{uu} (een reguliere onderdriehoeksmatrix), U^{uu} (een reguliere bovendriehoeksmatrix) en U^{uv} (een rechthoekige matrix) zodat geldt $\Phi_{,q} \sim L^{uu} \cdot [U^{uu}:U^{uv}]$, $\Phi_{,u} \sim L^{uu} \cdot U^{uu}$, $\Phi_{,v} \sim L^{uu} \cdot U^{uv}$ en $D = -U^{uu-1} \cdot U^{uv}$.

Een computeralgoritme voor het verkrijgen van geschikte beginvoorwaarden is te vinden in [1]. Het vinden van geschikte beginvoorwaarden kan een probleem opleveren bij systemen met veel vrijheidsgraden.

Het GCP-algoritme is een numeriek integratie algoritme met positieve-fouten contrôle. Door het gebruik van een predictor-corrector algoritme met variabele orde- en stapgrootte, waarbij alleen de onafhankelijke variabelen geintegreerd worden, kunnen de afhankelijke variabelen effectief berekend worden.

Verder kan er ook nog een onderscheidt gemaakt worden tussen het gebruik van Eulerhoeken en Eulerparameters bij 3D-analyse. Het gebruik van Eulerhoeken kan soms resulteren in numerieke moeilijkheden, als één of meerdere stijve lichamen aan grote rotaties worden blootgesteld. Als de 2e Euler rotatiehoek gelijk is aan $k\pi$ komen de assen van de 1e en 3e rotatiehoek overeen en kunnen deze 2 hoeken niet meer eenduidig bepaald worden. Daarom wordt meestal het gebruik van Eulerparameters geprefereerd boven het gebruik van Eulerhoeken [4].

2.3 Voor- en nadelen van de GCP-methode.

Het blijkt uit het voorgaande dat deze methode gebruikt maakt van de LU-decompositie. Het voordeel hiervan is dat het eenvoudig toepasbaar is en de uiteindelijke uitkomst goed hanteerbaar is voor integratie. Het GCP-algoritme is betrouwbaar en nauwkeurig. Het algoritme voldoet voor constraints aan de nauwkeurigheid die wordt gespecificeerd door de gebruiker en handhaaft goede fouten contrôle. Verder is de computertijd sterk gereduceerd t.o.v. bestaande integratiemethoden [2].

De GCP kan soms leiden tot slecht geconditioneerde matrices. Als dit gebeurt, is het noodzakelijk om een nieuwe set van onafhankelijke coördinaten te kiezen door het LU-decompositie proces te herhalen. Deze procedure vergroot niet alleen de bewerkingstijd, maar integratiefouten worden ook voortgeplant.

Er is wel een "nadeel" verbonden aan het integratie-algoritme. Bij het numeriek integratie-algoritme zijn de onafhankelijke coördinaten v bekend en worden de constraintvgl. opgelost voor de afhankelijke coördinaten u. Omdat de constraints niet-lineaire algebraïsche vgl. zijn, moeten iteratieve methoden gebruikt worden. Deze vereisen een schatting voor u in iedere tijdstap. De schatting mag niet te veel verschillen van de correcte oplossing, omdat anders divergentie van de oplossing kan optreden [3].
D.w.z. dat de effectiviteit van het algoritme begrensd is.

Ook zijn er verschillende factoren die de keuze van onafhankelijke coördinaten beïnvloeden.
- **De keuze van het eenhedensysteem.**
Omdat niet alle elementen van de Jacobi-matrix dezelfde fysische dimensie hebben, kunnen de numerieke waarden andere verhoudingen vormen, als verschillende eenhedenstelsels gebruikt worden. Je krijgt dan een ander pivotingproces, en dus ook verschillende afhankelijke en onafhankelijke coördinaten.
- **Type van pivoting.**
Partial (kolom) pivoting kan een ander resultaat opleveren dan full pivoting. Matrixbewerking met full pivoting kan voordeel hebben, t.a.v. de numerieke fout, t.o.v. partial pivoting.
- **Methode van matrixbewerking.**
Als een LU-decompositie wordt gebruikt voor een constraint Jacobi-matrix, i.p.v. een standaard Gaussische eliminatie, kunnen verschillende afhankelijke en onafhankelijke coördinaten verkregen worden.

2.4 Conclusie.

Al met al kan gezegd worden dat partitionering van gegeneraliseerde coördinaten m.b.v. de LU-decompositie een goede methode is voor het reduceren van een gekoppeld stelsel van DAE's. De effectiviteit van het integratie algoritme is begrensd door het gebruik van iteratieve methoden.

HOOFDSTUK 3

De Constraint Violation Stabilization methode (CVS)

3.1 Inleiding.

De CVS-methode is een alternatieve methode voor het oplossen van constrained mechanische systemen. Basisidee van deze methode is het uitdempen van fouten in constrainedvergelijkingen, door gebruik te maken van een gemodificeerde versnellingsvgl., die fouten in de constraintvgl. (1.2.5) en snelheidsvgl. (1.2.7) omvat.

De CVS-methode is een uitbreiding van de theorie van terugkoppelingscontrôle toegepast op de dynamische analyse van mechanische systemen. Het doel bij het ontwerpen van een terugkoppelingscontroller is het onderdrukken van foutengroei en het bereiken van een stabiele respons.

Systemen die beschreven worden door een 2e-orde differentiaalvgl. zoals

$$\ddot{y} = 0 \tag{3.1.1}$$

zijn instabiel, omdat een verstoring van buitenaf zoals ruis (of numerieke fouten i.g.v. een numeriek integratieproces) versterkt kan worden, d.w.z. het kan leiden tot waarden van $\dot{y}$ en y die ver van 0 liggen.
(3.1.1) wordt een <u>open-loop</u> systeem genoemd.

Daarentegen is een <u>closed-loop</u> systeem, zoals (3.1.2)

$$\ddot{y} + 2\alpha\dot{y} + \beta^2 y = 0 \tag{3.1.2}$$

stabiel als α en β^2 positieve constanten zijn met $\beta \neq 0$.

De termen $2\alpha\dot{y}$ en $\beta^2 y$ zijn de terugkoppelings contrôletermen die stabiliteit van de differtiaalvergelijking waarborgen.

3.2 Toepassen CVS-methode op de gekoppelde DAE's.

Het gekoppeld stelsel DAE's wordt door (3.2.1) beschreven.

$$\begin{bmatrix} M & \Phi_{,q}^T \\ \Phi_{,q} & 0 \end{bmatrix} \begin{bmatrix} \ddot{q} \\ \lambda \end{bmatrix} = \begin{bmatrix} Q^A \\ \gamma \end{bmatrix} \quad met \quad \gamma = -(\Phi_{,q}\dot{q})_{,q}\dot{q} - 2\Phi_{,qt}\dot{q} - \Phi_{,tt} \tag{3.2.1}$$

Voor resp. de constraintvgl., de snelheidsvgl. en versnellingsvgl. gelden de volgende uitdrukkingen:

$$\begin{aligned} \Phi &\equiv \Phi(q) = 0 \\ \dot{\Phi} &\equiv \Phi_{,q}\dot{q} = 0 \\ \ddot{\Phi} &\equiv \Phi_{,q}\ddot{q} - \gamma = 0 \end{aligned} \tag{3.2.2}$$

N.a.v. het voorgaande kan de numerieke oplossing van de 3e vgl. van (3.2.2) instabiel zijn en gaat deze vergelijking dus over in de gemodificeerde versnellingsvgl.

$$\ddot{\Phi} + 2\alpha\dot{\Phi} + \beta^2\Phi = 0 \tag{3.2.3}$$

met $\alpha \geq 0$, $\beta \neq 0$ en $\dot{\Phi} \approx \Phi \approx 0$, zodat deze vergelijking wel stabiel is. Hierin representeert Φ de fout in de constraintvgl. en $\dot{\Phi}$ de fout in de snelheidsvgl.
Door vergelijk van de 3e vgl. van (3.2.2) en (3.2.3) volgt nu dat de 2e vgl. van (3.2.1) geschreven kan worden als

$$\Phi_{,q}\ddot{q} = \gamma - 2\alpha(\Phi_{,q}\dot{q} + \Phi_{,t}) - \beta^2\Phi \equiv \gamma^* \tag{3.2.4}$$

Dit levert dan een gemodificeerd stelsel van gekoppelde DAE's, nl.:

$$\begin{bmatrix} M & \Phi_{,q}^T \\ \Phi_{,q} & 0 \end{bmatrix} \begin{bmatrix} \ddot{q} \\ \lambda \end{bmatrix} = \begin{bmatrix} Q^A \\ \gamma^* \end{bmatrix} \tag{3.2.5}$$

waarbij $\ddot{q}$ de berekende versnellingen representeert. Als er geen fout in de constraints zit wordt (3.2.5) gelijk aan (3.2.1).

Met deze toevoeging, bestaat het CVS integratie algoritme enkel uit het uitvoeren van directe integratie. Deze benadering is meer stabieler en nauwkeuriger dan het elementaire directe integratie algoritme, en is even snel qua computertijd [1]. De snelheid hangt sterk af van de keuze van α en β.

3.3 Evaluatie van de CVS-methode.

Er zijn enkele redenen aan te wijzen waarom deze methode niet in elke situatie nauwkeurige resultaten oplevert en qua computertijd voldoende efficiëntie bereikt.

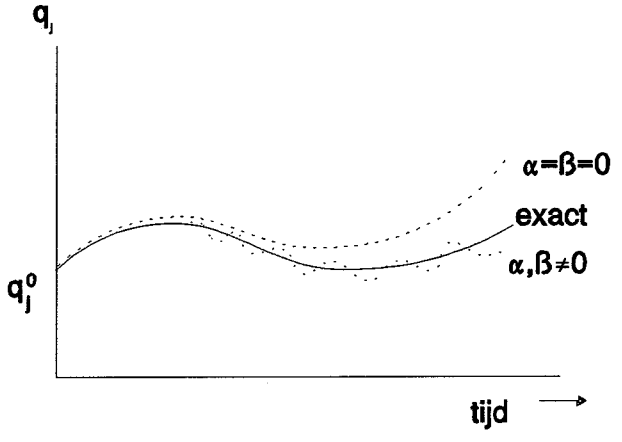
In de omgeving van kinematische singulariteiten kan de oplossing van de 3e vgl. van (3.2.2) wel een goede gedrag vertonen, maar een kleine verstoring in de rechterleden van (3.2.4) kan tot grote fouten leiden in de oplossing en dit leidt dan tot divergentie van het CVS-algoritme. Dit geeft geen goede oplossing meer of de tijdstap moet drastisch gereduceerd worden om fouten te vermijden [3].

Ook is er nog geen algemene methode gevonden om α en β te kiezen. Het effect van het introduceren van de terugkoppelingstermen is geillustreerd in figuur 2.

- als α en β beiden de waarde 0 wordt gegeven, kan het numerieke resultaat divergeren van de exacte oplossing.

- voor waarden van α en β ongelijk aan 0, oscilleert de oplossing rond de exacte oplossing. De amplitude en de frequentie van de oscillatie t.g.v. stabilizeringstermen hangen af van de waarden voor α en β. Experimenten hebben aangetoond dat voor veel practische problemen, een bereik voor de waarden α en β tussen 1 en 10 voldoende is.

- als $\alpha = \beta$ wordt kritische demping bereikt, zodat de responsie meestal sneller stabilizeert.



figuur 2. Schematische representatie van de exacte en numerieke opl. van een typische dynamische responsie

HOOFDSTUK 4

Een Hybride numerieke integratie methode.

4.1 Inleiding.

In dit hoofdstuk wordt een efficiënte en stabiele methode voor het oplossen van een set van gekoppelde differentiaal en algebraïsche vgl. (DAE's) voor een constrained dynamische systeem beschreven. Het numeriek integratie algoritme combineert de Constraint Violation Stabilization methode en de Gegeneraliseerde Coördinaten Partitionerings methode met elkaar, door gebruik te maken van resp. het voordeel van hun attractieve snelheden en fouten contrôle karakteristieken.

Uit de beschouwing van de GCP-methode bleek dat deze methode de DAE's impliciet reduceert naar onafhankelijke dv's voor numerieke integratie, waarbij het mogelijk wordt om positieve fouten contrôle toe te passen. De GCP handhaaft een zeer goede nauwkeurigheid, maar de efficiëntie is gelimiteerd door het gebruik van iteratieve oplossingen van de niet-lineaire algebraïsche constraintvgl. tijdens iedere integratietijdstap.

Uit de beschouwing van de CVS-methode bleek dat deze methode gebaseerd is op het toevoegen van termen aan de kinematische versnellingsvgl. om de positie- en snelheidsvgl. te stabilizeren. Alle gegeneraliseerde coördinaten worden dan gezien als onafhankelijk en worden geintegreerd door gebruik te maken van standaard numerieke integratie technieken. Deze methode is qua berekeningstijd snel, maar men kan geen positieve fouten contrôle onderhouden.

Het Hybride algoritme gebruikt de voordelen van beide methoden.

4.2 Het Hybride numerieke integratie algoritme.

Het algoritme volgt het basisidee van de GCP zover, dat er een set van onafhankelijke gegeneraliseerde coördinaten gedefinieerd wordt, gebaseerd op decompositie van de constrained Jacobiaan $\Phi_{,q}$.

Het algoritme gebruikt de gemodificeerde versnellingsvgl. (3.2.4) tijdens integratie, om kleine amplitude oscillaties t.g.v. constraintfouten te stabilizeren.

Constraintfouten worden gecontroleerd en de correctiestap bij het GCP-algoritme die de constraintvgl. voor de afhankelijke coördinaten $\dot{u}$ en u oplost, wordt alleen gebruikt als fouten in de positie- en snelheidsconstraints een gegeven foutentolerantie overschrijden.

Om instabiliteiten te voorkomen, die kunnen optreden bij het CVS-algoritme in de buurt van kinematische singulariteiten waardoor de coëfficiënten matrix

$$\begin{bmatrix} M & \Phi_{,q}^{T} \\ \Phi_{,q} & 0 \end{bmatrix}$$

slecht geconditioneerd kan worden, gaat het algoritme terug naar zuivere GCP.

Dus het Hybride-algoritme handhaaft de betrouwbaarheid en de positieve fouten contrôle karakteristieken van het GCP-algoritme en benaderd de rekensnelheid die gehaald kan worden met het CVS-algoritme.

Uit het algortime blijkt dat deze methode het beste resultaat geeft met een geoptimaliseerde keuze van de constanten α en β, maar tot nu toe is er nog geen algemene methode voor de keuze van α en β. Het algoritme gebruikt kleine vaste waarden van α en β om een behoorlijk deel van de constraintfouten uit te dempen en om zodoende de ongunstige invloed van de correctietermen op het integratieproces te voorkomen.
Dit algoritme is ook geimplementeerd in DADS.

De beschrijving van het algoritme is te vinden in [2] en [3].

4.3 Evalutie van de Hybride itegratie methode.

Het algoritme werkt in het kort als volgt:
Eerst wordt er gereduceerd door de GCP-methode, daarna kan men directe integratie toepassen met constraint stabilization en dan kan de iteratie gestart worden met een eventuele nieuwe partitionering van de coördinaten.

Uit experimenten blijkt dat de beschreven Hybride methode de gewenste voordelen van de GCP-methode en CVS-methode, met name de efficiëntie en nauwkeurigheid, behelst [3].

Als een te analyseren systeem zich rond een stabiele positie bevindt, gedraagt de methode zich als de meer efficiënte CVS-methode.
Is de beweging van het te analyseren systeem belangrijk, gedraagt de methode zich als de meer stabiele GCP-methode.

De Hybride methode handhaaft een grotere stapgrootte dan de CVS- en GCP-methode, en door contrôle van de integratiefout van alleen de onafhankelijke gegeneraliseerde coördinaten, elimineert de methode overbodige overhead in de predictor-corrector subroutine van het GCP-algoritme.

HOOFDSTUK 5

De Singuliere Waarden Decompositie (SWD)

5.1 Inleiding.

Overeenkomend met de tot nu toe besproken methoden gaat men bij deze methode ook uit van hetzelfde stelsel gekoppelde DAE's dat er uitziet als

$$\begin{bmatrix} M & \Phi_{,q}^T \\ \Phi_{,q} & 0 \end{bmatrix} \begin{bmatrix} \ddot{q} \\ \lambda \end{bmatrix} = \begin{bmatrix} Q^A \\ -(\Phi_{,q}\dot{q})_{,q}\dot{q} - 2\Phi_{,qt}\dot{q} - \Phi_{,tt} \end{bmatrix} \tag{5.1.1}$$

De conventionele methoden voor het oplossen van gewone differentiaalvgl. zijn over het algemeen niet toepasbaar op DAE's. De SWD is, net zoals de voorheen besproken LU-decompositie, wel geschikt om de bewegingsvgl. voor een klasse van constrained dynamische systemen te reduceren naar hun minimale dimensie.
De methode is gebaseert op decompositie van de constrained Jacobi-matrix $\Phi_{,q}$.

Bij het gebruik van de SWD worden niet alle n gegeneraliseerde coördinaten geintegreerd, maar slechts de n-m onafhankelijke coördinaten, waarna de afhankelijke coördinaten uit de constraintvgl. volgen.

Bij het gebruik van de SWD maakt men gebruik van een coördinatentransformatie via een constante orthogonale transformatiematrix als volgt:

$$z = Vq$$

Deze orthogonale matrix V wordt gepartitioneerd in een afhankelijke deelmatrix V^D, en een onafhankelijke deelmatrix V^I.

Als onafhankelijke coördinaten worden lineaire combinaties van de fysische gegeneraliseerde coördinaten q geselecteerd als

$$z^I = V^I q \tag{5.1.2}$$

waarin V^I een $(n-m)xn$ constante transformatiematrix is. Doordat de rijen van V^I onderling othogonaal gekozen worden, zijn de resulterende lineaire combinaties onderling onafhankelijk.

5.2 Basiseigenschappen van de SWD.

De mxn Jacobiaan $\Phi_{,q}$, met $m<n$, kan worden gedecomponeerd in de vorm

$$\Phi_{,q}=U^T D V \tag{5.2.1}$$

met U en V orthonormale matrices van resp. dimensie mxm en nxn. De mxn matrix D heeft de speciale vorm

$$D=\begin{array}{c} \left[\begin{array}{cccc|c} \epsilon_1 & & & & \\ & \epsilon_2 & & \emptyset & \\ & & \cdot & & \emptyset \\ & \emptyset & & \cdot & \\ & & & \epsilon_m & \end{array}\right] \\ \begin{array}{cc} m & n-m \end{array} \end{array} \tag{5.2.2}$$

De laatste $n-m$ kolommen van D zijn nullen en de ϵ_i's heten de singuliere waarden van matrix $\Phi_{,q}$, zo gerangschikt dat

$$\epsilon_1 \geq \epsilon_2 \geq \epsilon_3 \geq \ldots \geq \epsilon_m \geq 0$$

De singuliere waarden zijn positief en niet gelijk aan 0 als de matrix volle rang heeft.
Uit (5.2.1) volgt

$$\Phi_{,q}\Phi_{,q}^T = U^T D V V^T D^T U = U^T D D^T U = U^T \Lambda U \tag{5.2.3}$$

met Λ de diagonaalmatrix

$$DD^T = diag[\epsilon_1^2, \epsilon_2^2, \ldots, \epsilon_m^2]$$

Dus er geldt

$$\Phi_{,q}\Phi_{,q}^T U^T = U^T \Lambda U U^T = U^T \Lambda \tag{5.2.4}$$

met U^T de orthonormale modale matrix van $\Phi_{,q}\Phi_{,q}^T$. Dus de kolommen van U^T (de rijen van U) zijn orthonormale eigenvectoren van de symmetrische matrix $\Phi_{,q}\Phi_{,q}^T$ en de singuliere waarden ϵ_i zijn gelijk aan de wortels van de eigenwaarden van $\Phi_{,q}\Phi_{,q}^T$.

Analoog geldt

$$\Phi_{,q}{}^T\Phi_{,q}=V^TD^TUU^TDV=V^TD^TDV=V^T\Omega V \tag{5.2.5}$$

met Ω de diagonaalmatrix

$$D^TD=diag[\epsilon_1{}^2,\epsilon_2{}^2,\ldots,\epsilon_m{}^2\vdots 0\ldots 0]$$

De laatste n-m elementen op de diagonaal zijn nullen.
Analoog geldt dus ook

$$\Phi_{,q}{}^T\Phi_{,q}V^T=V^T\Omega VV^T=V^T\Omega \tag{5.2.6}$$

met V^T de orthonormale modale matrix van $\Phi_{,q}{}^T\Phi_{,q}$. Dus de kolommen van V^T (de rijen van V) zijn orthonormale eigenvectoren van de symmetrische matrix $\Phi_{,q}{}^T\Phi_{,q}$ en de singuliere waarden $\epsilon_i{}^2$ gevolgd door n-m nullen de corresponderende eigenwaarden.

5.3 Partitionering van de geg. coördinaten m.b.v. SWD.

Een nieuwe variabele z wordt gedefinieerd als

$$z=Vq \rightarrow q=V^Tz \tag{5.3.1}$$

Deze orthogonale transformatie levert een nieuwe vector z met gegeneraliseerde coördinaten voor het systeem.
De eerste tijdsafgeleide van (5.3.1) geeft (V is constant)

$$\dot{z}=V\dot{q} \tag{5.3.2}$$

De tweede tijdsafgeleide van (5.3.1) levert

$$\ddot{z}=V\ddot{q} \tag{5.3.3}$$

Beschouw nu een verstoring δz van z die voldoet aan de rvw $\Phi(z)=0$. M.b.v (5.3.1) wordt dit:

$$\Phi_{,q}\delta q=\Phi_{,q}V^T\delta z=0 \tag{5.3.4}$$

Uit (5.2.1) en het feit dat V orthonormaal is, resulteert

$$U^TD\delta z=0 \rightarrow D\delta z=0 \tag{5.3.5}$$

Omdat U orthonormaal is, mag men (5.3.5) voorvermenigvuldigen met U en gebruik maken van de vorm van D, resteert dan

$$[\epsilon_1 \delta z_1 \;\; \epsilon_2 \delta z_2 \;\; . \;\; . \;\; . \;\; \epsilon_m \delta z_m]^T = 0 \tag{5.3.6}$$

Dit toont aan dat $\delta z_{m+1} \ldots . \delta z_n$ niet berekend kunnen worden uit (5.3.5), en dus alleen uit de differentiaalvgl. van beweging. Daarom worden $z_{m+1} \ldots . z_n$ geselecteerd als de onafhankelijke gegeneraliseerde coördinaten voor het oplossen van de bewegingsvgl. en $z_1 \ldots . z_m$ als afhankelijke gegeneraliseerde coördinaten die berekend moeten worden uit de kinematische constrained vgl. Omdat de kinematische constraintvgl. niet-lineair zijn, is een iteratieve procedure, zoals de Newton-Raphson iteratie, nodig om tot een oplossing te komen. De set van onafhankelijke gegeneraliseerde coördinaten vertoont numerieke stabiliteit en een grote nauwkeurigheid voor een lange periode van simulatietijd, bij vergelijk van de SWD-decompositie t.o.v. de LU-decompositie, omdat de SWD-decompositie een nauwkeuriger algoritme is dan de LU-decompositie.

Als we nu alleen rvw onafhankelijk van de tijd beschouwen, dus $\Phi_{,t}=0$, dan krijgt men

$$\dot{\Phi} \equiv \Phi_{,q} \dot{q} = 0 \tag{5.3.7}$$

Voorvermenigvuldigen met U en gebruik maken van de definitie van z in (5.3.2) levert

$$D\dot{z} = 0 \tag{5.3.8}$$

Vanwege de speciale vorm van D is dit

$$[\epsilon_1 \dot{z}_1 \;\; \epsilon_2 \dot{z}_2 \;\; . \;\; . \;\; . \;\; \epsilon_m \dot{z}_1]^T \tag{5.3.9}$$

Aangezien de ϵ_i's niet nul zijn voor een Jacobiaan met rang m, dus volle rang, betekent dit

$$[\dot{z}_1 \;\; \dot{z}_2 \;\; . \;\; . \;\; . \;\; \dot{z}_m]^T = 0 \tag{5.3.10}$$

Omdat orthonormale transformaties norm behouden, dus

$$||\dot{z}|| = ||\dot{q}|| \tag{5.3.11}$$

en omdat $z_1 \ldots . z_m$ nul zijn, volgt nu

$$\sum_{i=m+1}^{n} \dot{z}_i^2 = \sum_{i=1}^{n} \dot{q}_i^2 \tag{5.3.12}$$

Voor een systeem met eenheidsmassa's en -traagheidsmomenten ($M=I$) geeft (5.3.12) aan dat de kinetische energie van de onafhankelijke samengestelde coördinaten de totale kinetische energie omvat. Dit levert dus een criterium om vast te stellen wanneer een nieuwe set van onafhankelijke samengestelde coördinaten moet worden gedefinieerd.

Uit (5.3.1) en (5.3.2) kunnen de virtuele verplaatsingsvector δq en snelheidsvector $\dot{q}$ uitgedrukt worden in termen van V als:

$$\begin{aligned} \delta q &= \sum_{j=m+1}^{n} \delta z_j V_j^T \\ \dot{q} &= \sum_{j=m+1}^{n} \dot{z}_j V_j^T \end{aligned} \qquad (5.3.13)$$

Dit betekent dat de verplaatsingen beperkt worden tot een deelruimte van R^n die wordt opgespannen door $V_{m+1}^T \ldots V_n^T$ en dat snelheden langs de $V_1^T \ldots V_m^T$-assen nul zijn.
Dit houdt in dat de onafhankelijke samengestelde coördinaten alle systeeminformatie bevatten. Matrix V kan nu gepartitioneerd worden in twee submatrices V^I en V^D, die resp. de onafhankelijke en afhankelijke gedeelten van matrix V voorstellen:

$$V = \begin{bmatrix} V^{DT} \\ \cdots \\ V^{IT} \end{bmatrix}^T \begin{matrix} (m) \\ \\ (n-m) \end{matrix} \qquad (5.3.14)$$

De wijze van opsplitsing van de orthogonale matrix V in de submatrices V^I en V^D hangt af van de manier waarop de gegeneraliseerde coördinaten q gepartioneerd worden in afhankelijke en onafhankelijke coördinaten.

Nadat de onafhankelijke samengestelde posities en snelheden zijn berekend, kunnen de fysische coördinaten en snelheden berekend worden met matrixvgl. van de vorm

$$\begin{bmatrix} \Phi_{,b} \\ \cdots \\ V^I \end{bmatrix} x = b$$

5.4 Evaluatie van de SWD.

In het kort komt de SWD-methode hierop neer:
De m onafhankelijke samengestelde coördinaten z^I zijn lineaire combinaties van de n fysische coördinaten q volgens (5.1.2). Hierin is V^I een mxn matrix die uit de m constraintvgl. wordt bepaald. Enkel de onafhankelijke coördinaten hoeven geintegreerd worden. De fysische coördinaten kunnen weer hieruit worden berekend. Bij het integreren van z^I blijkt het systeem zich langs een raaklijn aan de constraints te bewegen [7],[8].

5.5 Conclusies.

Gegeneraliseerde coördinaten partitionering met SWD genereert de n-m dimensionale tangentiële subruimte van de kinematische constraints waarin de beweging plaats vindt.

Het blijkt ook, dat de onafhankelijke samengestelde snelheden alle informatie bevatten die is opgeslagen in de fysische snelheidsvector. De coëfficientenmatrix betrokken bij oplossingen voor fysische posities, bij gebruik van Newton-Raphson iteratie, en de oplossingen van de fysische snelheden uit de snelheid constraints zijn goed geconditioneerd als de Jacobi systeemmatrix ook goed geconditioneerd is.
Dit resulteert in nauwkeurige oplossingen van de bew.vgl.

Bovendien, integratie van de onafhankelijke samengestelde coördinaten benadert alle fysische coördinaten langs het tangentiële constraint oppervlak, wat zorgt voor goede beginschattingen voor de Newton-Raphson integratie.

De onafhankelijke samengestelde coördinaten blijken een betere set van onafhankelijke coördinaten voor numerieke oplossingen van de bewegingsvgl. te zijn, dan iedere andere set van onafhankelijke fysische coördinaten.
Dit laat toe dat tijdens het integratie-algoritme grote tijdstappen genomen kunnen worden wat de berekeningstijd van de computer aanzienlijk reduceert, o.g.v. het feit dat de SWD-decompositie een numeriek nauwkeuriger algoritme is dan de LU-decompositie [9].

HOOFDSTUK 6

De Projectie methode

6.1 Inleiding.

De projectie-methode is een alternatieve methode om de differentiële bewegingsvgl. af te leiden voor klassieke mechanismen. Deze benadering kan gebruikt worden om dezelfde bewegingsvgl. af te leiden die ook afgeleidt kunnen worden bij het gebruik van de Lagrangevgl.

M.b.v. Lagrange worden de vgl. meestal afgeleid door het gebruik van virtuele formalisering, bijv. het concept van virtuele verplaatsingen en virtuele arbeid. Alhoewel dit een efficiënte en betrouwbare mathematische techiek is, ontbreekt vaak het (geometrische) inzicht bij de op te lossen problemen. Bovendien, de methoden gericht op automatische eliminatie van constrained krachten, beantwoorden meestal niet aan het probleem hoe men de constraint krachten moet bepalen.

De voordelen van deze methode zijn de gemakkelijkheid van afleiding, z'n intuitieve karakter, het fysische inzicht van de methode en het feit dat deze methode algemener van aard is dan de Lagrangevergelijkingen. Voor een uitgebreide beschrijving van deze methode ,een generalisering o.g.v. een uitgebreide wiskundige achtergrond, zie **Appendix A.**

De projectie-methode is gebaseerd op de projectie van de initiële (constraint krachten omvattende) dynamische vergelijkingen op de orthogonale en de tangentiële deelruimte; de orthogonale deelruimte wordt opgespannen door de constraintvectoren, en de tangentiële deelruimte complementeert de orthogonale deelruimte in de configuratieruimte van het systeem.
De tangentiële projectie geeft de belasting-vrije (of pure kinetische) bewegingsvgl., terwijl de orthogonale projectie de constraint krachten bepaalt.
In termen van de vectorruimte betekent dit, dat we de wet van Newton projecteren op een deelruimte.

In de volgende paragrafen zal de algemene opzet van deze methode besproken worden. Voor een beschrijving met holonome en/of niet-holonome constraints wordt verwezen naar [11] en [12], waarbij [12] gedeeltelijk opgenomen is in **Appendix A.** Een vergelijkbare beschrijving van de uitwerking a.h.v. holonome constraints zal in **§6.3** besproken worden, waarin ook de verschillen in de keuze van onafhankelijke gegeneraliseerde coördinaten besproken zal worden a.h.v. een vergelijk met de projectie methode, de SWD en de GCP met LU-decompositie.

Verder zal ook een numeriek algoritme besproken worden dat gebaseerd is op de afleiding in §6.3.

6.2 Introductie van de projectiemethode.

De algemene opzet van de methode zoals ze in het volgende beschreven wordt is gebaseerd op een artikel van Scott [10].

De toestand van een systeem kan beschreven worden door x, een vector met carthesische coördinaten, of door q, een vector met gegeneraliseerde coördinaten als (zie ook **Appendix AA1**):

$$x=\begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ x_n \end{bmatrix} \quad , \quad q=\begin{bmatrix} q_1 \\ q_2 \\ . \\ . \\ q_m \end{bmatrix} \tag{6.2.1}$$

De relatie tussen deze 2 coördinatensystemen, waarbij geldt dat $n \geq m$, wordt gegeven door

$$x=x(q,t) \tag{6.2.2}$$

Vgl. (6.2.2) 1x differentiëren levert

$$\dot{x}=J\dot{q}+b \tag{6.2.3}$$

Nogmaals differentiëren levert

$$\ddot{x}=J\ddot{q}+\dot{J}\dot{q}+\dot{b} \tag{6.2.4}$$

Hierin is J de Jacobiaan van de carthesische coördinaten x t.o.v. de gegeneraliseerde coördinaten q, en b de partiële afgeleide van x t.o.v de tijd, als

$$J=\begin{bmatrix} \frac{\partial x_1}{\partial q_1} & . & \frac{\partial x_1}{\partial q_m} \\ . & . & . \\ \frac{\partial x_n}{\partial q_1} & . & \frac{\partial x_n}{\partial q_m} \end{bmatrix} \quad , \quad b=\begin{bmatrix} \frac{\partial x_1}{\partial t} \\ . \\ \frac{\partial x_n}{\partial t} \end{bmatrix}$$

De bewegingsvgl. kunnen dan geschreven worden als

$$F=M\ddot{x} \tag{6.2.5}$$

Hierin is F de vector van krachten die werkt op de lichamen in het systeem en staat M voor de massamatrix.

Als we nu (6.2.4) substitueren in (6.2.5) krijgt men

$$F=M(J\ddot{q}+\dot{J}\dot{q}+\dot{b}) \tag{6.2.6}$$

Het idee achter de projectie methode is als volgt:

Relatie (6.2.2) houdt in dat de positie van het systeem, dus de vector x, voorgeschreven is en op een deelruimte van R^n ligt. Deze deelruimte wordt beschreven via (6.2.2), door de m parameters $q_1, q_2, \ldots, q_m$, en de tijd t. Omdat het systeem beperkt is tot deze deelruimte, betekent dit dat alle krachten die niet werken in een richting tangentieël aan deze deelruimte, ook geen effect hebben op de beweging van het systeem.

Bij een gegeven vector x op deze deelruimte (dus gegeven $x=x(q,t)$ in R^m) wordt de ruimte van alle richting die raken aan deze deelruimte van x de **tangentiële deelruimte** genoemd. Waar we nu naar toe willen is dat we de orthonormale projectie van het rechterlid en het linkerlid van (6.2.6) op de tangentiële ruimte willen bepalen.
Opgemerkt dient te worden dat de kolommen van J lineair onafhankelijk zijn. De kolommen van de Jacobiaan J spannen dus de tangentiële deelruimte op, omdat de kolommen van J bestaan uit de partiële afgeleiden van de positievector x t.o.v. de verschillende gegeneraliseerde coördinaten.
De tangentiële deelruimte is de set van alle vectoren die gerepresenteerd kunnen worden in de vorm Jw voor elke $w \epsilon R^n$.

Definieer $\mathbb{P}\{z\}$ als de orthogonale projectie van de vector z op de tangentiële deelruimte. Als resultaat krijgen we dan:

$\mathbb{P}\{z\}=0$ als dan en slechts dan geldt dat $J^T z=0$.

<u>Bewijs</u>: stel $\{j_1, j_2, \ldots, j_n\}$ is een orthonormale basis voor de tangentiële deelruimte. Als de orthogonale projectie van $z=0$, geldt

$$(z,j_1)\,j_1+(z,j_2)\,j_2+\ldots+(z,j_n)\,j_n=0$$

waarbij (.,.) inwendinge producten voorstellen. Omdat de j's lineair onafhankelijk zijn, geldt dus ook dat $(z,j_i)=0$.
Hieruit volgt dat vector z loodrecht staat op iedere vector in de tangentiële deelruimte. Dit kan ook algebraïsch uitgedrukt worden door $(z,Jw)=0$ voor elke $w \epsilon R^n$. Er geldt dus ook $(J^T z,w)=0$ voor elke w. Hieruit volgt dus dat $J^T z=0$.

Nu nemen we dus de orthogonale projectie van (6.2.6) op de tangentiële deelruimte, als

$$P_T[F-M(J\ddot{u}+\dot{J}u+\dot{b})]=0 \tag{6.2.7}$$

Hieruit resulteert dan

$$J^T[F-M(J\ddot{u}+\dot{J}\dot{u}+\dot{b})]=0 \Rightarrow$$

$$J^TF=J^TMJ\ddot{u}+J^TM\dot{J}\dot{u}+J^TM\dot{b} \qquad (6.2.8)$$

Vgl. (6.2.8) is nu de bewegingsvgl. verkregen door de projectie methode voor systemen, waarvan de bewegingsvgl. geformuleerd kunnen worden in de vorm van (6.2.5).
Voor een eenvoudige toepassing van de projectie methode a.h.v. bovenstaande besproken uitwerking zie [10]. Hierin wordt een slinger als voorbeeld behandelt.

6.3 De projectie methode voor systemen met holonome constraints.

6.3.1 Inleiding.

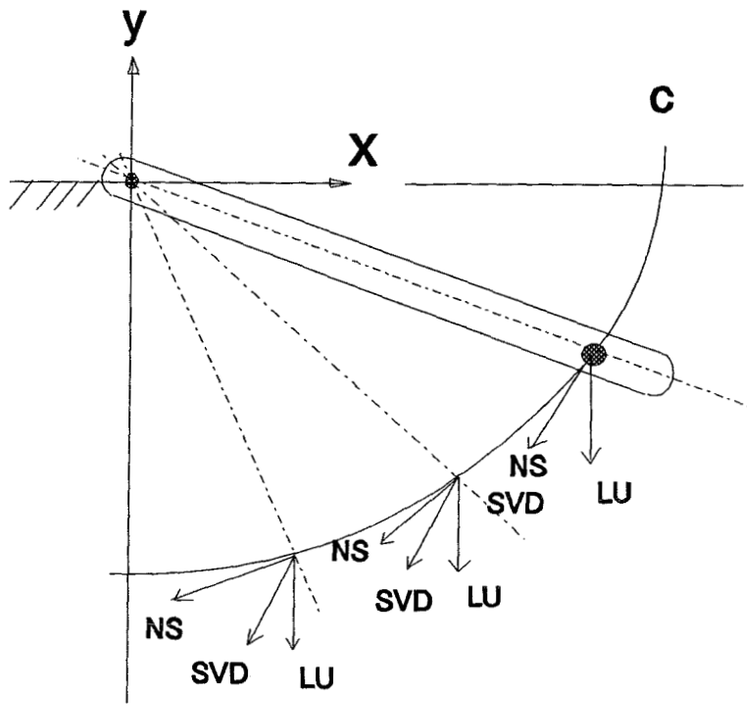
Hier zal, net zoals in de voorgaande hoofdstukken het gekoppeld stelsel DAE's centraal staan, dus met holonome constraints. Deze differentiële algebraïsche vergelijkingen zien er als volgt uit:

$$\begin{bmatrix} M & \Phi_{,q}^T \\ \Phi_{,q} & 0 \end{bmatrix}\begin{bmatrix} \ddot{q} \\ \lambda \end{bmatrix}=\begin{bmatrix} Q^A \\ \gamma \end{bmatrix} \quad met \quad \gamma=-(\Phi_{,q}\dot{q})_{,q}\dot{q}-2\Phi_{,qt}\dot{q}-\Phi_{,tt} \qquad (6.3.1)$$

In de volgende paragraaf zal een continue en differentieerbare basis van de constraint ruimte, de **Null Space** (**NS**) geintroduceerd worden, die automatisch gegenereerd wordt door het Gram-Schmidt proces. De onafhankelijke coördinaten worden verkregen door een transformatie van de fysische snelheidscoördinaten op de tangentiële deelruimte van het constraintoppervlak.
Het resultaat hiervan is dat de onafhankelijke coördinaten op het constraintoppervlak liggen en dus geen constraint violation control nodig is [13].
Doordat een orthogonale matrix V gegenereerd en tevens gepartitioneerd wordt en daarna een coördinatentransformatie plaats vindt, wordt een set van onafhankelijke coördinaten automatisch gegenereerd, wat de efficiëntie van het numeriek algoritme verbetert. (het numeriek algortime zal nog ter sprake komen).

In de voorgaande hoofdstukken zijn een aantal methoden behandelt die ook een set van onafhankelijke coördinaten genereren, nl. de SWD en de GCP met LU-decompositie. Deze 2 methoden en de NS-methode (de projectie methode) kunnen met elkaar vergeleken worden m.b.t. de keuze voor de onafhankelijke coördinaten na een transformatie van de gegeneraliseerde coördinaten.

De verschillende keuze van de onafhankelijke coördinatenvector kan geillustreerd worden aan de hand van een eenvoudige slinger die zich in een plat vlak bevindt (figuur 3).



figuur 3. Verschillende keuze van de onafhankelijke coördinaten voor een slinger.

Het constrained oppervlak van het systeem is de cirkel **c**, geillustreerd in de figuur. Het massamiddelpunt van de slinger beweegt over deze cirkel. Omdat dit een systeem is met één vrijheidsgraad, is de basis van de constraint null space de tangentiële vector rakend aan de cirkel, en deze is eenduidig bepaald.
In het begin kiest de SWD- en de NS-methode dezelfde onafhankelijke coördinatenvector, omdat deze tangentiële vector eenduiding bepaald is voor het systeem. De LU-decompositie methode kiest de verticale coördinaat als de onafhankelijke coördinaat.
Tijdens simulatie, worden de onafhankelijke coördinatenvectoren van de LU-decompositie en de SWD-methode constant gehouden totdat voorgeschreven criteria overschreden worden. Als dit gebeurt zal men constraint violation control moeten toepassen.
De onafhankelijke coördinatenvector van de NS-methode blijft in de constraint null space, zijn richting blijft tangentieël aan de constraint oppervlak, en bevat alle snelheidsinformatie voor het systeem.
In het artikel van Liang en Lance [13] wordt een vergelijk gemaakt tussen de projectiemethode en de LU-decompositie.
Het blijkt dat de projectiemethode veel nauwkeuriger is gebaseerd op het criterium van energiebehoud.

6.3.2 Het Gram-schmidt proces en de NS.

Het Gram-Schmidt proces genereert automatisch de **null space** (**NS**) van de constraint ruimte. De berekende basis van de NS is differentieërbaar en eenduiding als de constraints continue zijn (dus ook differentieërbaar).
Uitgangspunt van het Gram-Schmidt is dat er een nxn matrix P wordt geconstrueerd die is samengesteld uit de nxm getranspo-neerde Jacobiaan $\Phi_{,q}^T$ en een $nx(n-m)$ boolean matrix B^T als

$$P=[\Phi_{,q}^T \;\vdots\; B^T] \tag{6.3.2}$$

De Jacobiaan matrix heeft volle rang en de boolean matrix B is willekeuring met als eis dat de matrix P regulier moet zijn. De matrix B kan bepaald worden uit een SWD, een LU-decomposi-tie of door de gebruiker gespecificeerde begincondities.

Het Gram-Schmidt proces transformeert de matrix P naar een orthogonale matrix V, waarbij V gepartitioneerd wordt in de submatrices V_D en V_I met resp. dezelfde grootte als de matrices $\Phi_{,q}^T$ en B^T, als

$$V=[V_D \;\vdots\; V_I] \tag{6.3.3}$$

NB.: de orthogonale matrix V wordt gepartioneerd in de subma-trices V_I en V_D. Omdat de submatrix V_I gebruikt wordt voor een snelheidstransformatie, waarbij de gegeneraliseerde snelheden omgezet worden in onafhankelijke snelheden, krijgt deze matrix V_I als index I(independent).

Het Gram-Schmidt proces, een recursief algoritme, ziet er als volgt uit:

$$v_i=a_i\{p_i-\sum_{j=1}^{i-1}(v_j^T p_i)\,v_j\} \qquad i=1,\ldots,n$$

$$a_i=1/(\|p_i-\sum_{j=1}^{i-1}(v_j^T p_i)\,v_j\|)$$

waarbij de gevraagde norm de Euclidische 2-norm voorstelt. Bijv., bij een vector betekent de Euclidische 2-norm :

$$\|x\|=\sqrt{x^T x}$$

De orthonormale set vectoren v_i zijn de kolommen van de ortho-gonale matrix V.

Bij het algoritme wordt ook gebruik gemaakt van de afgeleide van de orthogonale matrix V.

Deze wordt als volgt berekend:

$$\dot{v}_i = \dot{a}_i \{p_i - \sum_{j=1}^{i-1} (v_j^T p_i)\, v_j\} + a_i \{\dot{p}_i - \sum_{j=1}^{i-1} [\,(\dot{v}_j^T p_i + v_j^T \dot{p}_i)\, v_j + (v_j^T p_i)\, \dot{v}_j]\,\}$$

$$met \quad \dot{a}_i = -\frac{[\dot{p}_i^T p_i - \sum_{j=1}^{i-1} (v_j^T p_i)\,(\dot{v}_j^T p_i + v_j^T \dot{p}_i)\,]}{[\,(p_i^T p_i) - \sum_{j=1}^{i-1} (v_j^T p_i)^2\,]^{\frac{3}{2}}}$$

Na partitionering van V zijn de vectoren van de matrix V_D lineaire combinaties van de vectoren van de matrix $\Phi_{,q}^T$, dus V_D en $\Phi_{,q}^T$ spannen dezelfde deelruimte op.
Omdat V_I orthogonaal is t.o.v. V_D, is V_I ook orthogonaal t.o.v. $\Phi_{,q}^T$ en kan de matrixvorm van deze relatie gegeven worden als

$$\Phi_{,q} V_I = 0 \qquad (6.3.4)$$

Dus, **de null space van de constraint Jacobiaan $\Phi_{,q}^T$ wordt gedefinieerd door V_I.**

Deze beschrijving van het Gram-Schmidt proces, zoals het verwerkt moet worden in het algoritme voor de projectiemethode is ook terug te vinden in [13].
Voor verdere informatie van het Gram-Schmidt proces wordt verwezen naar de boeken van E.Haug en Tchung [14], Shampine en Gordon [15] en James W. Longley [16].

6.3.3 Berekenen van de onafhankelijke coördinaten en de gemodificeerde versnellingsvergelijking.

Bij het bepalen van van de onafhankelijke coördinaten wordt de snelheidsvector getransformeerd i.p.v de positievector. Dit is wenselijk omdat de snelheidsvector op de constraint tangentiële deelruimte ligt en de positievector niet.
Er wordt na deze transformatie van de snelheidsvector een set van onafhankelijke coördinaten gepakt die op het constraint oppervlak liggen en niet alleen op de tangentiële deelruimte.

Verder blijkt dat de Lagrange multipliers in (6.3.1) geëlimineerd kunnen worden door de transformatie m.b.v. het Gram-Schmidt proces, omdat de constraint krachten het systeem op het constraintoppervlak houden en verder geen enkele invloed hebben op de beweging van het systeem.

Recapitulerend: een dynamisch systeem wordt gekarakteriseerd door een n-vector van gegeneraliseerde coördinaten q en m constraints als

$$\Phi(q) = 0 \tag{6.3.5}$$

De constraints worden verondersteld holonoom en tijdsonafhankelijk te zijn.
De Lagrange multiplier formulering geeft de bewegingsvgl.

$$M\ddot{q} + \Phi_{,q}{}^{T}\lambda = Q^{A} \tag{6.3.6}$$

Voor de eerste en tweede tijdsafgeleide van (6.3.5) geldt

$$\begin{aligned} \Phi_{,q}\dot{q} &= 0 \quad \ldots(1) \\ \Phi_{,q}\ddot{q} + (\Phi_{,q}\dot{q})_{,q}\dot{q} &= 0 \quad \ldots(2) \end{aligned} \tag{6.3.7}$$

Omdat de Jacobiaan $\Phi_{,q}$ de constraint deelruimte van het systeem definieert, definieert de null space van de constraint deelruimte de tangentiële deelruimte van de constraint deelruimte. D.m.v het Gram-Schmidt proces wordt een orthogonale matrix V gegenereert. Deze orthogonale matrix V wordt gebruikt als een transformatiematrix om een nieuwe n-snelheidsvector $\dot{z}$ te definieren als

$$\dot{z} = V^{T}\dot{q} \tag{6.3.8}$$

Omdat V orthogonaal en regulier is, bestaat er een 1 op 1 relatie tussen $\dot{q}$ en $\dot{z}$ en wordt de vectornorm in stand gehouden, alsmede de orthogonaliteitseis

$$\dot{q} = V\dot{z} \tag{6.3.9}$$

Nu wordt $\dot{z}$ op dezelfde manier gepartioneerd als matrix V. Definieer $\dot{z}_D$ en $\dot{z}_I$ als de afhankelijke en onafhankelijke termen van de nieuwe snelheidsvector met resp. als dimensies mx1 en $(n-m)$x1. De snelheidsvector $\dot{z}$ en de versnellingsvector $\ddot{z}$ worden dan geschreven als

$$\dot{z} = \begin{bmatrix} \dot{z}_D \\ \cdots \\ \dot{z}_I \end{bmatrix}, \quad \ddot{z} = \begin{bmatrix} \ddot{z}_D \\ \cdots \\ \ddot{z}_I \end{bmatrix} \tag{6.3.10}$$

In **Appendix C** wordt duidelijk gemaakt dat alle snelheidsinformatie, die nodig is voor de beweging van het systeem, wordt behelst door de onafhankelijke snelheid $\dot{z}_I$.
Geometrisch wil dit zeggen, dat men de snelheidsvector $\dot{z}_I$ op de tangentiële deelruimte van het constraintoppervlak legt en $\dot{z}_D$ is dan een vector loodrecht op dat oppervlak.
Omdat de normaalvector nul is (zie **Appendix C**), wordt de beweging beperkt tot het constraintoppervlak, en treedt er geen constraint violation op.

Uit (4) en (12) van **Appendix C** blijkt dus dat de variabele $\dot{z}_I$ op het constraintoppervlak ligt, en niet alleen op de constraint tangentiële deelruimte. Als de elementen van $\dot{z}_I$ gekozen worden als de onafhankelijke coördinaten van het constrained dynamisch systeem, beschreven door (6.3.1), zal de numerieke oplossing tot tevredenheid stemmen, zonder dat er constraint violation stabilization (CVS; hoofdstuk 3) of constraint violation controle nodig is, wat wel het geval is als men het constraint dynamisch systeem m.b.v SWD of LU-decompositie wil reduceren.

Zoals eerder aangeduidt, is het mogelijk om de Lagrange multipliers te elimineren uit (6.3.6).

In het voorgaande was afgeleid dat de submatrix V_I de null space van de constraint Jacobiaan opspant.
Als we nu (6.3.6) voorvermenigvuldigen met de matrix V_I^T krijgen we

$$V_I^T M\ddot{q} + V_I^T \Phi_{,q}^T \lambda = V_I^T Q^A \tag{6.3.11}$$

Omdat $\Phi_{,q}^T$ en V_I orthogonaal zijn, is de tweede term in het linkerlid van (6.3.11) gelijk aan nul en dit houdt in dat de Lagrange multipliers gelijk aan nul zijn. Er resteert dan

$$V_I^T M\ddot{q} = V_I^T Q^A \tag{6.3.12}$$

Subtitutie van (6.3.9) na partitionering in (6.3.12) geeft

$$V_I^T M(V_D\ddot{z}_D + V_I\ddot{z}_I + \dot{V}_D\dot{z}_D + \dot{V}_I\dot{z}_I) = V_I^T Q^A \tag{6.3.13}$$

Omdat $\dot{z}_D$ en $\ddot{z}_D$ beiden gelijk aan nul zijn krijgt men als resulterende vergelijking:

$$V_I^T M V_I \ddot{z}_I + V_I^T M \dot{V}_I \dot{z}_I = V_I^T Q^A \tag{6.3.14}$$

Vgl. (6.3.14) is nu de bewegingsvgl. op het constraintoppervlak in termen van de onafhankelijke coördinaten z_I.

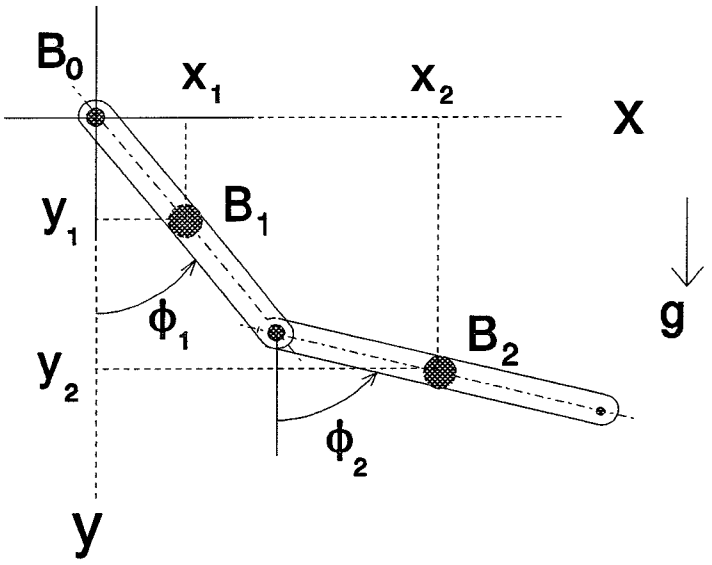
Voor het numeriek algoritme wordt (6.3.14) omgeschreven in een andere vorm voor de bepaling van de versnellingen:

$$\ddot{z}_I = \tilde{M}^{-1} G \dot{z}_I + \tilde{M}^{-1} V_I^T Q^A$$
$$\tilde{M} = V_I^T M V_I \,;\; G = -V_I^T M \dot{V}_I \qquad (6.3.15)$$

De eerste term in het rechterlid van (6.3.15) representeert de loodrechte versnelling die afhangt van constraint geometrie en de snelheid en de tweede term is de tangentiële versnelling.

6.4 Algoritme voor de projectie methode.

Hier wordt een algoritme beschreven om de DAE's van een constrained dynamisch systeem op te lossen, gebaseerd op de in §6.3 verkregen resultaten en [13].
Bij het programmeren van het algortime wordt de dubbelslinger in figuur 4 als voorbeeld genomen.



figuur 4. Dubbelslinger in 2 dimensies.

Hierin hebben de lichamen $\mathbf{B_1}$ en $\mathbf{B_2}$ dezelfde lengte **l=0.5m**.
Het blijkt dat als men de bewegingsvgl. van deze dubbelslinger analytisch berekent men te maken krijgt met 6 differentiaalvergelijkingen van de vorm

$$M\ddot{q} + \Phi_{,q}^T \lambda = F$$

en 4 constraintvergelijkingen (dus dimensie van Φ:m=4).

In matrixvorm resteert dan voor de 6 differentiaalvgl.:

$$\begin{bmatrix} J_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & m_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & m_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & J_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & m_2 & 0 \\ 0 & 0 & 0 & 0 & 0 & m_2 \end{bmatrix} \cdot \begin{bmatrix} \ddot{\phi}_1 \\ \ddot{x}_1 \\ \ddot{y}_1 \\ \ddot{\phi}_2 \\ \ddot{x}_2 \\ \ddot{y}_2 \end{bmatrix} +$$

$$+ \begin{bmatrix} -\frac{1}{2}l\cos(\phi_1) & \frac{1}{2}l\sin(\phi_1) & l\cos(\phi_1) & -l\sin(\phi_1) \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{2}l\cos(\phi_2) & -\frac{1}{2}l\sin(\phi_2) \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ m_1 g \\ 0 \\ 0 \\ m_2 g \end{bmatrix}$$

Voor de constraintmatrix $\Phi(q)$ geldt dan:

$$\Phi(q) = \begin{bmatrix} x_1 - \frac{1}{2}l\sin(\phi_1) \\ y_1 - \frac{1}{2}l\cos(\phi_1) \\ l\sin(\phi_1) - (x_2 - \frac{1}{2}l\sin(\phi_2)) \\ l\cos(\phi_1) - (y_2 - \frac{1}{2}l\cos(\phi_2)) \end{bmatrix} = 0$$

Het is duidelijk dat men nu gebruik maakt van een reductie-methode om deze vgl. gemakkelijk op te lossen , waarbij men een "ontkoppelt" stelsel vergelijkingen krijgt, die m.b.v. numerieke integratie gemakkelijker zijn op te lossen.
Na reductie krijgt men 2 niet-lineaire differentiaalvgl., waarbij voor $\mathbf{B}_1$ het volgende geldt:

$$\begin{cases} (J_1 + \frac{1}{4}m_1 l^2 + m_2 l^2)\ddot{\phi}_1 + \frac{1}{2}m_2 l^2 \ddot{\phi}_2 \cos(\phi_1 - \phi_2) + \\ + \frac{1}{2}m_2 l^2 \dot{\phi}_2^{\,2} \sin(\phi_1 - \phi_2) + g l \sin(\phi_1)\,[\frac{1}{2}m_1 + m_2] = 0 \end{cases}$$

Voor $\mathbf{B}_2$ geldt:

$$\begin{cases} (J_2 + \frac{1}{4} m_2 l^2)\ddot{\phi}_2 + \frac{1}{2} m_2 l^2 \ddot{\phi}_1 \cos(\phi_1 - \phi_2) - \\ -\frac{1}{2} m_2 l^2 \dot{\phi}_1^2 \sin(\phi_1 - \phi_2) + \frac{1}{2} m_2 g l \sin(\phi_2) = 0 \end{cases}$$

Er wordt nu geprobeerd om deze vergelijkingen m.b.v. het algoritme voor de projectiemethode op te lossen.

In het voorgaande werd duidelijk dat men een coördinaten transformatie toepast om een onderscheidt te maken tussen afhankelijke en onafhankelijke coördinaten.
Omdat de orthogonale coördinaten transformatie plaats vindt in het snelheidsdomein, is het moeilijk om de positievector te beperken tot het constraintoppervlak. Daarom worden de snelheidsvector $\dot{q}$ en de onafhankelijke versnellingsvector $\ddot{z}_I$ geintegreerd om de positievector q en de onafhankelijke snelheidsvector $\dot{z}_I$ te krijgen.

<u>Het algoritme is als volgt:</u>

Het tijdsinterval dat wordt beschouwd is (t_0, t_{end}) en index i duidt het huidige tijdstip aan, d.w.z. i=0 betekent $t=t_0$.

stap 1: Lees de beginpositie, -snelheid en andere systeemdata. Construeer de Jacobiaan $\Phi_{,q}$ en een boolean matrix B uit een SWD, een LU-decompositie of door de gebruiker gespecificeerde begincondities.
Corrigeer de positievector q^0 door het gebruik van Newton-Raphson iteratie ((6.4.1),(6.4.2)) met de n-m beginposities r^0 en snelheden $\dot{r}^0$ waarbij de gebruiker aangeeft welke posities en snelheden volgens hem accuraat zijn.

$$\begin{bmatrix} \Phi_{,q_k}{}^0 \\ B \end{bmatrix} \Delta q^0{}_k = \begin{bmatrix} -\Phi(q^0{}_k) \\ -Bq^0{}_k + r^0 \end{bmatrix} \tag{6.4.1}$$

$$q^0{}_{k+1} = q^0{}_k + \Delta q^0{}_k \ , \quad k=1,2,\ldots. \tag{6.4.2}$$

Hierin is k een iteratieteller. Bij iedere berekening van Δq_0 worden Φ en $\Phi_{,q}$ opnieuw berekend. (6.4.1) berekent achtereenvolgende correcties voor q_k^0 totdat aan alle kinematische constraints voldaan is met de gewenste nauwkeurigheid.

Bereken de beginsnelheidsvector $\dot{q}^0$ door gebruik van dezelfde coëfficientenmatrix als in (6.4.1) als

$$\begin{bmatrix} \Phi_{,q}{}^0 \\ B \end{bmatrix} \dot{q}^0 = \begin{bmatrix} 0 \\ \dot{r}^0 \end{bmatrix} \tag{6.4.3}$$

stap 2: Gebruik het Gram-Schmidt proces zoals in **§6.3.2**, voer de transformatie uit van de getransponeerde van de coëfficientenmatrix in vgl. (6.4.1) naar de orthogonale matrix $\boldsymbol{V}$. Identificeer de $\boldsymbol{V}_D$ en $\boldsymbol{V}_I$ submatrices door partitioneren van de orthogonale matrix $\boldsymbol{V}$.

$$\left[\Phi_{,q}{}^{0^T} \;\vdots\; B^T\right] \rightarrow \underline{Gram\text{-}Schmidtproces} \rightarrow \left[V_D{}^0 \;\vdots\; V_I{}^0\right] \tag{6.4.4}$$

De onafhankelijke beginsnelheidsvector wordt gevormd uit

$$\dot{z}_I{}^0 = V_I{}^{0^T} \dot{q}^0 \tag{6.4.5}$$

Matrix B wordt hergedefinieerd als

$$B = V_I{}^{0^T} \tag{6.4.6}$$

Dus B is orthogonaal t.o.v. $\Phi_q{}^{oT}$.

stap 3: Bepaal de tijdsafgeleide van de Jacobiaan matrix

$$\frac{\partial}{\partial t}(\Phi_{,q}{}^i) = (\Phi_{,q}{}^i \dot{q}^i)_{,q^i} \tag{6.4.7}$$

en bereken de tijdsafgeleide van $\boldsymbol{V}_I^i$, $\dot{\boldsymbol{V}}_I^i$.

stap 4: Bereken de n-m onafhankelijke versnellingscoördinaten, gebaseert op vgl. (6.3.15) als

$$\ddot{z}_I{}^i = \tilde{M}^{-1} G \dot{z}_I{}^i + \tilde{M}^{-1} V_I{}^{i^T} Q^{A(i)} \tag{6.4.8}$$

De gegeneraliseerde versnellingsvector $\ddot{q}$ kan gevonden worden uit

$$\ddot{q}^i = V_I{}^i \ddot{z}_I{}^i + \dot{V}_I{}^i \dot{z}_I{}^i \tag{6.4.9}$$

stap 5: Integreer $(\dot{q}^i, \ddot{z}_I^i)$ met $(q^0, \dot{z}_I^0)$ als begincondities om $(q^{i+1}, \dot{z}_I^{i+1})$ op $t=t_{end}$ te krijgen, m.b.v. een variabele orde en variabele stapgrootte integratie routine.
Hiertoe wordt een subroutine gestart die het interval (t_0, t_{end}) verdeelt in deelintervallen Δt_i zodat $t_{i+1}=t_i+\Delta t_i$.
Bij elke integratiestap worden de volgende stappen afgewerkt:

5a: Bereken d.m.v. integratie $(q^{i+1}, \dot{z}_I^{i+1})$ met als begincondities $(q^0, \dot{z}_I^0)$. De index i wordt nu 1 opgehoogd zodat $t=t_{i+1}=t_i+\Delta t_i$.

5b: Voer het Gram-Schmidt proces uit

$$\left[\Phi_{,q}^{i+1^T} \;\vdots\; B^T\right] \rightarrow \underline{Gram\text{-}Schmidtproces} \rightarrow \left[V_D^{i+1} \;\vdots\; V_I^{i+1}\right] \qquad (6.4.10)$$

Herdefinieer zonodig matrix B als $B=(V_I^{i+1})^T$.

5c: Bereken de gegeneraliseerde snelheidsvector $\dot{q}$ als

$$\dot{q}^{i+1} = V_I^{i+1} \dot{z}_I^{i+1} \qquad (6.4.11)$$

5d: Bepaal de tijdsafgeleide van de Jacobiaan matrix

$$\frac{\partial}{\partial t}(\Phi_{,q}^{i+1}) = (\Phi_{,q}^{i+1} \dot{q}^{i+1})_{,q^{i+1}} \qquad (6.4.13)$$

en bereken de tijdsafgeleide van V_I^{i+1}, $\dot{V}_I^{i+1}$, zoals in **§6.3.2.**

5e: Bereken de n-m onafhankelijke versnellingscoördinaten als

$$\ddot{z}_I^{i+1} = \tilde{M}^{-1} G \dot{z}_I^{i+1} + \tilde{M}^{-1} V_I^{i+1^T} Q^{A(i+1)} \qquad (6.4.14)$$

5f: De gegeneraliseerde versnellingsvector $\ddot{q}$ wordt gevormd uit

$$\ddot{q}^{i+1} = V_I^{i+1} \ddot{z}_I^{i+1} + \dot{V}_I^{i+1} \dot{z}_I^{i+1} \qquad (6.4.15)$$

5g: Dit proces wordt herhaald. Indien $t_{i+1}=t_{end}$ wordt de subroutine beëindigd, zoniet worden de stappen **5a** t/m **5f** herhaald.

Dit algoritme wordt geprogrammeerd in Fortran-taal(F77).

6.5 Toelichting bij het algoritme.

Het algoritme uit §6.4 is verwerkt in een fortanprogramma, waarvan de listing te vinden is in **Appendix D**. Hier zal de werking ervan in het kort worden uitgelegd.

6.5.1 Uitleg van het programma.

Allereerst worden de dimensies van het probleem vastgesteld: het aantal lichamen NB en het aantal constraints worden aangegeven. Vervolgens worden begin- en eindtijdstip ingelezen. Dan wordt stap 1 van het algoritme uitgevoerd.
Vectoren worden in het programma voorgesteld als ééndimensionale arrays, matrices door tweedimensionale arrays. Zo wordt de gegeneraliseerde coördinaat q aangeduid met Q(I), $\dot{z}_I$ met ZIP(I), $\dot{q}$ met QP(I), de boolean matrix B met B(I,J) etc. Om verwarring te voorkomen wordt de vector met gegeneraliseerde krachten Q^A aangeduid met QU(I) en de massamatrix M met MA(I,J).
Verder wordt het Newton-Raphson proces uitgevoerd door CALCQ (berekening van de beginpositie) en CALCQP (berekening van de beginsnelheid). Hierin zijn de vgl. (6.4.1) en (6.4.2) verwerkt.

Bij elke iteratie worden FI(I)(=Φ) en FQ(I,J)(=Φ_q) opnieuw berekend, door het aanroepen van de subroutine CALFIQ. De berekening van Q(I) wordt stopgezet als de lengte van DQ(I) (=Δq) kleiner is dan 0.1×10^{-3}. Vervolgens wordt de definitieve FIQ(I,J) berekend door nogmaals subroutine CALFIQ aan te roepen.

Daarna wordt stap 2 uitgevoerd. Het gebruik van het Gram-Schmidt proces om de Null Space te genereren en de matrix VI(I,J)(=V_I) te definieren, m.b.v subroutine VORT. Vervolgens wordt de onafhankelijke snelheidsvector ZIP(I) berekend, m.b.v subroutine CALZIP en wordt matrix B hergedefinieerd.

Vervolgens worden de tijdsafgeleiden van de Jacobiaan, subroutine CGAMMA, en van de matrix VI(I,J), VIP(I,J)(=$\dot{V}_I$), subroutine CALVIP, berekend.(stap 3).

Het berekenen van de onafhankelijke versnellingscoördinaten ZIPP(I) (stap 4) wordt uitgevoerd door subroutine CZIPP, waarna de gegeneraliseerde versnellingen QPP(I) berekend worden door subroutine CALQPP.

Het uitvoeren van stap 5 omvat het vaststellen van de begincondities Y(I), gevolgd door het aanroepen van de integratiesubroutine DO2CAF.(De beschrijving van deze subroutine en andere gebruikte subroutines uit de NAG-library is te vinden in de **bijlage**). Na elke integratiestap roept deze subroutine op zijn beurt de zelfgeschreven subroutine FCN aan.

Deze subroutine FCN werkt de volgende onderdelen af:

5a: toekennen van de berekende waarden voor Y(I) aan Q(I) en ZIP(I)

5b: uitvoeren van het Gram-Schmidt proces voor het opnieuw berekenen van VI(I,J) m.b.v subroutine VORT

5c: berekenen van de gegeneraliseerde snelheidsvector QP(I) door subroutine CALQPI

5d: opnieuw berekenen van de tijdsafgeleiden van de Jacobiaan door subroutine CGAMMA en van matrix VI(I,J), VIP(I,J) door subroutine CALVIP

5e: ZIPP(I) wordt opnieuw berekend door subroutine CZIPP

5f: QPP(I) wordt opnieuw berekend door subroutine CALQPP

5g: herhalen van deze integratie totdat T0=TEND, waarna subroutine D02CAF beeindingd is, en de resultaten opgeborgen worden in een uitvoerfile voor weergave op het scherm.

In de diverse subroutines zijn de dimensies aangeven van de vectoren en matrices, die actief zijn in de desbetreffende subroutines.

Nogmaals dient opgemerkt te worden dat het algoritme geschreven is a.h.v. het voorbeeld van een dubbelslinger. Dit levert een niet-lineaire set van bewegingsvergelijkingen met holonome constraints. Als men dit programma voor andere problemen wil gebruiken dan moeten er verscheidene wijzingen in aangebracht worden; verandering van dimensies resulteert in verandering van de dimensies van vectoren en matrices, andere constraints etc.

6.6 Conclusies.

Uit het voorgaande wordt het duidelijk dat de beschreven projectiemethode voor de analyse van constrained dynamische systemen een aantal voordelen heeft wat resulteert in compacte wiskundige formuleringen, gelijke behandeling van systemen met holonome en niet-holonome constraints[10], en het fysisch inzicht bij dynamische problemen. De methode verschaft bruikbaar meetkundig inzicht bij problemen, zodat de wiskundige transformaties duidelijker worden en ze meer begrijpelijker maakt.

Wat al eerder werd opgemerkt is dat de projectiemethode een andere manier is om de vergelijkingen van Lagrange af te leiden. Uit §6.2 kan men constateren dat de afleiding simpel en kort is.

Een groot voordeel van deze methode is dat men met deze methode systemen met holonome en niet-holonome constraints kan beschrijven, wat met de SWD-methode en de GCP-methode met LU-decompositie niet mogelijk is.

Zoals gezien is deze methode gebaseerd op het gebruik van het **Gram-Schmidt proces** om **onafhankelijke coördinaten** te genereren die continue en differentieerbaar zijn. Deze onafhankelijke coördinaten zijn gedefinieerd op constraintoppervlak zodat aan de constraints voldaan is zonder dat er constraint violation contrôle nodig is.

Het grote voordeel van de continuïteit van de onafhankelijke coördinaten is dat de onafhankelijke coördinaten ongevoelig zijn voor systeemveranderingen en zich er dus geen problemen zullen voordoen(zie ook figuur 3). Het ongevoeling worden van de onafhankelijke coördinaten levert wel problemen op bij de andere methoden.
Hier zullen de onafhankelijke coördinaten hergedefinieerd moeten worden. Dit gebeurt principieël tijdens simulaties wat resulteert in aanzienlijke veranderingen in de systeemconfiguratie.

Uit het voorgaande kan men aannemen dat de methode een hogere efficiëntie zal bereiken omdat de bepaling van de basis van de **Null Space** door het gebruik van het Gram-Schmidt proces alleen vector vermenigvuldiging met zich meebrengt.

APPENDIX A

A1. De Projectie methode: inleiding.

Deze beschrijving (gedetailleerd uitgelegt in [10]) is een uitbreiding van de beschrijving geintroduceerd door Scott [11].
Omdat in het volgende een algemene opzet van de projectiemethode besproken wordt, zal het startpunt van de hier beschreven analyse de resulterende vgl. van een niet voorgeschreven (unconstrained) systeem met n graden van vrijheid zijn. M.a.w., de analyse wordt uitgevoerd in de n-dimensionale (configuratie) ruimte, en de beweging van het systeem kan geinterpreteerd worden als de beweging van een gegeneraliseerd deeltje met n coördinaten. Als er m onafhankelijke constraints op het systeem verondersteld worden, zullen de samenhangende constraintvectoren een m-dimensionale deelruimte opspannen.

Verwijzend naar het voorbeeld van een deeltje dat beweegt over een glad oppervlak, noemt men de **m-deelruimte de orthogonale deelruimte**, en z'n **k-dimensionale ($k=n-m$) complement** in de n-dimensionale ruimte, **de tangentiële deelruimte**.

De projectie van de initiële dynamische vergelijkingen in de orthogonale en de tangentiële deelruimte is de sleutel van deze beschreven methode.

Zoals al gezegd, geeft de tangentiële projectie de constrained belasting-vrije (of pure kinetische) bewegingsvergelijkingen, en de orthogonale projectie dient voor het bepalen van de constrained krachten.

In de nu volgende bespreking van de projectiemethode zullen de gevallen van holonome en niet-holonome constraints apart behandelt worden, en de analyse wordt gestart vanuit de constraintvgl. die getransformeerd zijn naar een 2^e-orde kinematische vorm.

A2. Probleemformulering.

We bekijken mechanische systemen die gekarakteriseerd worden door n geg. coördinaten $x=[x_1,...,x_n]^T$. De bewegingsvgl. kunnen in de volgende matrixvorm geschreven worden als

$$\begin{cases} M\dot{v}=h^* \\ \dot{x}=Av+a_0 \end{cases} \tag{2.1}$$

Hierin is $M(x,t)$ een nxn symmetrische positief-definiete matrix, $v=[v_1,...,v_n]^T$ is een kolommatrix met kinematische parameters, $A(x,t)$ is een nxn inverteerbare (transformatie) matrix, $h^*(v,x,t)$ en $a_v(x,t)$ zijn $nx1$ matrices, t is de tijd.

Uit het oogpunt van de vectorruimte en tensoralgebra analyse, kan M geinterpreteerd worden als de metrische tensor matrix van de basis $e_v=[e_{v1},...,e_{vn}]^T$, dus $M=e_v e_v^T$; v en x zijn de covariante representaties van de vectoren $\boldsymbol{v}$ en $\boldsymbol{x}$ in de basis e_v en e_x, resp. $\boldsymbol{v}=v^T e_v$ en $\boldsymbol{x}=x^T e_x$; en h^* is de covariante representatie van de toegepaste centrifugale krachtvector in de basis e_v^*, $\boldsymbol{h}=h^{*T} e_v^*$. De aanduiding (*) betekent hier de covariante representatie van vectoren en contravariante basis vectoren, b.v. $e_v=Me_v^*$, en $\dot{v}^*=M\dot{v}$ ($\dot{v}^*$ duidt de representatie van traagheidskrachten aan).

Opgemerkt dient te worden dat alle positie-, snelheid- en versnellingsvectoren gerepresenteerd worden door contravariante componenten, terwijl de krachtvectoren gerepresenteerd worden door covariante componenten. Het onderscheid tussen contravariante en covariante componenten van vectoren staat beschreven in **Appendix AA2**.

De vergelijkingen van (2.1) zijn hier bedoelt als de initiële vergelijkingen van het niet voorgeschreven (unconstrained) systeem. Hier wordt niet alleen een systeem mee bedoelt dat een verzameling is van onbeperkte (unconstrained) deeltjes en/of lichamen, maar ook een systeem wiens dynamisch gedrag al eerder was geformuleerd in de onafhankelijke coördinaten door elke andere methode, bv. de responsievrije bewegingsvergelijkingen van een systeem met onderling verbonden lichamen.
De tweede vgl. van (2.1) beschrijft een transformatie tussen de geg. snelheden $\dot{x}$ en de kinematische parameters v die vaak geintroduceerd worden in practische toepassingen. De kinematische parameters zijn of <u>nieuwe geg. snelheden</u> als de componenten van

$$v=B\dot{x}+b_0 \tag{2.2}$$

met $B=A^{-1}$ en $b_0=-A^{-1}a_0$, integreerbaar zijn, of <u>quasi-snelheden</u> als de componenten niet integreerbaar zijn.

In het bijzonder: als $b_{i0}=0$ zal de i^e-component van $v_i=B^{(i)}\dot{x}$, met $B^{(i)}$ de i^e-rij van B, een nieuwe gegeneraliseerde coördinaat zijn als de Jacobiaan $\partial B^{(i)}/\partial x$ een symmetrische matrix is, bv. $\partial B_j^{(i)}/\partial x_k=\partial B_k^{(i)}/\partial x_j$, met $j,k=1,...,n$. In het andere geval zal v_i een quasi-snelheid zijn. Ter vereenvoudinging zal v de vector van quasi-snelheden genoemd worden.
Veronderstel nu dat m-onafhankelijke constraints voor het systeem gelden en introduceer de constraintvgl. in de 2^e-orde kinematische vorm

$$C_\lambda \dot{v}+c_{\lambda 0}^*=0 \tag{2.3}$$

met $C_\lambda(v,x,t)$ een mxn constraint matrix met maximale rang, zodanig dat de zgn. constraintvectoren worden opgeslagen in C_λ^T als kolommen; en $c_{\lambda 0}^*(v,x,t)$ is een mx1 matrix.
De kolommen van C_λ^T zijn de covariante representatie van de

constraintvectoren in de basis e_v^*.

Als er **geometrische (of holonome) constraints, $f(x,t)$** en/of **1e-orde kinematische (of niet-holonome) constraints $\phi(v,x,t)$** aanwezig zijn in een systeem, moeten ze getransformeerd worden in de vorm van (2.3) door resp. twee of één keer te differentiëren naar de tijd.

In dit geval resteren voor C_λ en $c_{\lambda 0}^*$ de volgende relaties:

$$C_\lambda = \begin{cases} \frac{\partial f}{\partial x} A \; ; \; f(x,t) = 0 \\ \frac{\partial \phi}{\partial v} \; ; \; \phi(v,x,t) = 0 \end{cases} \tag{2.4}$$

$$c_{\lambda 0}^* = \begin{cases} \left(\frac{\partial f}{\partial x} A\right)^{\cdot} v + \left(\frac{\partial f}{\partial x} a_0 + \frac{\partial f}{\partial t}\right)^{\cdot} \; ; \; f(x,t) = 0 \\ \frac{\partial \phi}{\partial x}(Av + a_0) + \frac{\partial \phi}{\partial t} \quad ; \; \phi(v,x,t) = 0 \end{cases} \tag{2.5}$$

Het is duidelijk, dat de transformatie van f=0 en ϕ=0 in de vorm van (2.3) geschikte condities inhoudt opgelegt aan de initiële waarden v_0 en x_0.
De constraints van (2.3) opgelegt voor het systeem introduceren responsies die moeten voorkomen in de 2e vgl. van (2.1). Hier wordt een verzameling van ideale constraints bekeken, wat inhoudt dat van de constraintresponsies geëist wordt dat ze collineair (op één lijn liggend) zijn met de samenhangende constraint vectoren. Daarom worden de constrained gegeneraliseerde krachten geldend voor het systeem ten gevolge van de constraints in (2.3) gerepresenteerd in de richtingen van v als volgt:

$$r^* = \sum_{i=1}^{m} r_i^* = \sum_{i=1}^{m} c_{\lambda i}^* \lambda = C_\lambda^T \lambda \tag{2.6}$$

met $\lambda = [\lambda_1, \ldots, \lambda_m]$ de vector van niet berekende multipliers, en $c_{\lambda i}$ de i^e-kolom van C_λ^T.

Dit leidt uiteindelijk tot een set van bewegingsvgl. voor een systeem met ideale constraints, bestaande uit $2n+m$ vgl.:

$$\begin{cases} M\dot{v} = h^* + C_\lambda^T \lambda & ..(a) \\ \dot{x} = Av + a_0 & ..(b) \\ C_\lambda \dot{v} + c_{\lambda 0}^* = 0 & ..(c) \end{cases} \tag{2.7}$$

A3. De Projectiemethode formulering.

Beschrijf de representatie van de kolommatrix van constraint-vectoren door $e_\lambda=[e_{\lambda 1},\ldots,e_{\lambda m}]^T$, dan zijn de covariante componenten in de basis e_v^* opgeslagen in C_λ^T als kolommen. e_λ is onafhankelijk (rang(C_λ)=m=maximaal), dus er bestaat een set van $k=n-m$ onafhankelijke vectoren $e_\tau=[e_{\tau 1},\ldots,e_{\tau k}]^T$, die orthogonaal is t.o.v. e_λ; e_τ wordt gerepresenteerd door contravariante componenten in de basis e_v, die zijn opgeslagen in C_τ^T als kolommen ($C_\tau(v,x,t)$ is een kxn matrix van maximale rang). T.g.v. orthogonaliteit geldt:

$$C_\tau C_\lambda^T=0 \leftrightarrow C_\lambda C_\tau^T=0 \tag{3.1}$$

d.w.z. C_τ is een orthogonaal complement van C_λ in de n-dim. ruimte. Verder representeert (3.1) ook de inwendige producten $e_\tau e_\lambda^T$ of $e_\lambda e_\tau^T$ van orthogonale vectoren.
Omdat de vectoren $e'=[e_\lambda^T,e_\tau^T]$ lineair onafhankelijk zijn, vormen ze een nieuwe basis in de n-dimensionale ruimte.
Er kan nu een formule van de transformatie van de basisvectoren geschreven worden als volgt:

$$e' = \begin{bmatrix} e_\lambda \\ e_\tau \end{bmatrix} = \begin{bmatrix} C_\lambda M^{-1} \\ C_\tau \end{bmatrix} e_v = T e_v \tag{3.2}$$

met e_v en e' covariante basisvectoren.

Omdat de dynamische vgl. (2.7a) worden gerepresenteerd in covariante componenten, is de (covariante) representatie van de vgl. in de basis e'^* gelijk aan de vermenigvuldiging van de linkse termen in (2.7a) met de transformatiematrix T.

Uit het feit dat e_λ **een orthogonale deelruimte van** e_v **opspant**, en e_τ **een tangentiële deelruimte van** e_v **opspant**, kunnen de resulterende dynamische vgl. in de basis e'^* geprojecteerd worden op de twee deelruimten als volgt:

$$\begin{bmatrix} C_\lambda M^{-1} \\ C_\tau \end{bmatrix} \left[M\dot{v} = h^* + C_\lambda^T \right]$$

$$\Rightarrow \begin{cases} C_\lambda \dot{v} = C_\lambda M^{-1} h^* + C_\lambda M^{-1} C_\lambda^T & ..(a) \\ C_\tau M \dot{v} = C_\tau h^* & ..(b) \end{cases} \tag{3.3}$$

Nu vormen (2.7b), (2.7c) en (3.3b) een nieuwe set van vergelijkingen die geschreven kunnen worden als

$$\begin{cases} TM\dot{v}=h'^* & ..(a) \\ \dot{x}=Av+a_0 & ..(b) \end{cases} \tag{3.4}$$

met $h'^*=[(-c_{\lambda 0}^*)^T,(C_r h^*)^T]^T$. De dimensie van (3.4) is nu gereduceerd tot $2n$ i.p.v. (2.7) met dimensie $2n+m$.
De orthogonale projectie (3.3a) kan dienen voor expliciete berekening van de niet berekende Lagrange-multipliers en dus ook voor de berekening van de constraintresponsies.
Met (2.7c) volgt uit (3.3a)

$$-c_{\lambda 0}^* = C_\lambda M^{-1} h^* + C_\lambda M^{-1} C_\lambda^T \lambda$$

$$\begin{aligned} \Rightarrow \quad \lambda &= -(C_\lambda M^{-1} C_\lambda^T)^{-1}\{c_{\lambda 0}^* + C_\lambda M^{-1} h^*\} \\ &= -M_\lambda^{-1}\{c_{\lambda 0}^* + C_\lambda M^{-1} h^*\} \end{aligned} \tag{3.5}$$

met $M_\lambda(v,x,t)=C_\lambda M^{-1} C_\lambda^T$ een mxm matrix (= een metrische tensor van de orthogonale deelruimte) en $\lambda=\lambda(v,x,t)$. De i^e constraintkracht kan gevonden worden uit

$$r_i^* = c_{\lambda i}^* \lambda_i = r_i^*(v,x,t) \tag{3.6}$$

met $c_{\lambda i}^*$ de i^e kolom van C_λ^T.

ALs de constraintvergelijkingen (2.3) (of vergelijking (2.7c)) de getransformeerde vormen van de lagere orde constraintvergelijking, kan directe integratie van (3.6) leiden tot constraintviolation t.g.v. numerieke integratiefouten.
Er zijn tenminste 2 fundamentele benaderingen met als doel het vermijden van constraint violation.
De 1e benadering is het gebruiken van speciale integratietechnieken gebaseerd op controleren van constraintviolation bij iedere integratiestap en gericht op reduceren van de violation, dus de methode minimaliseert in feite alleen de constraintviolation. De C.S.V. methode van Baumgarte kan als voorbeeld dienen.
De 2e benadering zijn methoden gebaseerd op het uitdrukken van de dynamische vergelijkingen in onafhankelijke coördinaten, zo gekozen dat aan alle constraintvergelijkingen is voldaan. (Bv. G.C.P. of Kane's vergelijkingen). Een bijkomend voordeel is reductie van de dimensie van de resulterende dynamische vergelijkingen.
Hier zal de tweede benadering besproken en uitgevoerd worden vanuit het oogpunt van de projectiemethode.
Voorafgaand aan de presentatie van deze formulering zullen enkele mathematische relaties geintroduceerd worden, welke

gebruikt worden bij verdere analyse.

De metrische tensormatrix M' van de basis $e'(=Te_v)$ is

$$M' = TMT^T = \begin{bmatrix} C_\lambda M^{-1} C_\lambda^T & 0 \\ 0 & C_\tau M C_\tau^T \end{bmatrix} = \begin{bmatrix} M_\lambda & 0 \\ 0 & M_\tau \end{bmatrix} \tag{3.7}$$

met M_λ en M_τ de metrische tensormatrices van resp. de orthogonale en tangentiële deelruimte. Uit (3.8) wordt duidelijk dat de deelruimten elkaar complementeren.
Uit (3.7) kan de inverse van de transformatiematrix T gevonden worden als

$$T^{-1} = MT^T(M')^{-1} = MT^T \begin{bmatrix} M_\lambda^{-1} & 0 \\ 0 & M_\tau^{-1} \end{bmatrix} \tag{3.8}$$

Definieer een vector van onafhankelijke quasi-versnellingen $\dot{u}=[\dot{u}_1,\ldots,\dot{u}_k]^T$ en kies ze zo dat ze niet gerepresenteerd worden in de orthogonale deelruimte. M.a.w., (2.3) kan gezien worden als een set van m quasi-versnellingen die t.g.v. de opgelegde constraints altijd gelijk aan 0 zijn.
<u>Daarom zijn er alleen maar $k=(n-m)$ onafhankelijke quasi-versnellingen</u>.
Wiskundig kan het geformuleerd worden als

$$M' \begin{bmatrix} 0 \\ \dot{u} \end{bmatrix} = \begin{bmatrix} 0 \\ M_\tau \dot{u} \end{bmatrix} = \begin{bmatrix} C_\lambda \dot{v} + c_{\lambda 0}^* \\ C_\tau M \dot{v} + c_{\tau 0}^* \end{bmatrix} = TM\dot{v} + \begin{bmatrix} c_{\lambda 0}^* \\ c_{\tau 0}^* \end{bmatrix} \tag{3.9}$$

met $c_{\tau 0}^*(v,x,t)$ een k-dimensionale vector.
Uit (3.9) volgt, rekening houdend met (3.7)

$$\dot{v} = T^T(M')^{-1}\left(M' \begin{bmatrix} 0 \\ \dot{u} \end{bmatrix} - \begin{bmatrix} c_{\lambda 0}^* \\ c_{\tau 0}^* \end{bmatrix}\right) =$$

$$\begin{aligned} &= C_\tau^T \dot{u} - (M^{-1} C_\lambda^T M_\lambda^{-1} c_{\lambda 0}^* + C_\tau^T M_\tau^{-1} c_{\tau 0}^*) \\ &= C_\tau^T \dot{u} + c_{u0} \end{aligned} \tag{3.10}$$

Door nu (3.10) te substitueren in (3.4a) resulteert

$$M' \begin{bmatrix} 0 \\ \dot{u} \end{bmatrix} - \begin{bmatrix} c_{\lambda 0}^* \\ c_{\tau 0}^* \end{bmatrix} = \begin{bmatrix} -c_{\lambda 0}^* \\ C_\tau h^* \end{bmatrix} \tag{3.11}$$

De 1e m-vergelijkingen van (3.11) geven de identiteit $-c_{\lambda 0}^{*}=-c_{\lambda 0}^{*}$ en de overblijvende k-vergelijkingen zijn de dynamische vergelijkingen uitgedrukt in de onafhankelijke quasi-versnellingen, gerepresenteerd in de tangentiële deelruimte:

$$M_{\tau}\dot{u}=C_{\tau}h^{*}+c_{\tau 0}^{*}=h_{\tau}^{*} \tag{3.12}$$

A4. De Projectiemethode geassocieerd met holonome constraints.

Veronderstel dat de m constraints opgelegd aan het systeem holonomische constraints zijn van de vorm

$$f(x,t)=0 \tag{4.1}$$

Van de constraints wordt verondersteld dat ze onafhankelijk zijn, dus (4.1) kan gebruikt worden als een set van m kromlijninge coördinaten, die t.g.v. de opgelegde constraints gelijk aan nul zijn op ieder moment van de beweging van het te analyseren systeem. (**Appendix AA1**)
Dan kan de positie van het systeem expliciet uitgedrukt worden door $k=(n-m)$ onafhankelijke coördinaten $q=[q_1,...,q_k]^T$.
Er bestaat, tenminste theoretisch, een inverse relatie die er uitziet als

$$x=x(q,t) \tag{4.2}$$

Introduceer nu een vector van onafhankelijke quasi-snelheden $u=[u_1,...,u_k]^T$ en definieer een relatie, corresponderend aan de 2e vergelijking van (2.1), bv.

$$\dot{q}=Du+d_0 \tag{4.3}$$

met $D(q,t)$ een kxk inverteerbare (transformatie) matrix en $d_0(q,t)$ een k-dimensionale vector.
M.b.v. substitutie en differentiatie van de vergelijkingen (4.3) en (3.4b) en (4.2) krijgt men

$$\begin{aligned}\dot{v}&=A^{-1}JD\dot{u}+(A^{-1}JD)^{\cdot}u+(A^{-1}(Jd_0+\frac{\partial x}{\partial t}-a_0))^{\cdot}\\&=A^{-1}JD\dot{u}+c_{u0}\end{aligned} \tag{4.4}$$

met $J=\partial q/\partial x$ de nxk Jacobiaan matrix. (Zie **Appendix AA3**).

Als men nu (3.10) vergelijkt met (4.4), is het duidelijk dat er geldt

$$C_\tau^T = A^{-1}JD \tag{4.5}$$

en met $A(x(q,t))$ kan men C_τ, gedefinieerd in (4.5), uitdrukken als een afhankelijkheid van q en t. Eveneens kan c_{u0} in (4.4) uitgedrukt worden als een functie van u, q en t.

Men kan nu afleiden dat C_τ gedefinieerd door (4.5) werkelijk een orthogonaal complement van C_λ, gedefinieerd door (2.4), is. De dynamische constraintconditie (2.3) houdt in $C_\lambda \delta\dot{v}=0$, en uit (4.5) volgt dat $\delta\dot{v}=A^{-1}JD\delta\dot{u}$, dus $C_\lambda(A^{-1}JD)\,\delta\dot{u}=0$. Omdat alle $\delta\dot{u}$ onafhankelijk zijn, resulteert dus $C_\lambda(A^{-1}JD)=0$. Hieruit volgt dus d.m.v (4.6) $C_\lambda C_\tau^T=0$, wat de orthogonaliteits-conditie uitdrukt.

Na substitutie van (4.5) in (3.4a), kunnen de gereduceerde dynamische vergelijkingen, gelijk aan (3.12), gevonden worden als

$$M_\tau \dot{u} = (A^{-1}JD)^T(h^* - M\,c_{u0}) = h_\tau^* \tag{4.7}$$

met

$$\begin{cases} M_\tau = (A^{-1}JD)^T M (A^{-1}JD) = C_\tau M C_\tau^T \\ -(A^{-1}JD)^T M c_{u0} = -C_\tau M c_{u0} = c_{\tau 0}^* \end{cases}$$

waarbij $c_{\tau 0}^*$ refereert aan (3.9) en (3.12).
Met $M_\tau = M_\tau(q,t)$ en $h_\tau = h_\tau(u,q,t)$, wordt de set van nieuwe bewegingsvgl., bestaande uit (4.4) en (4.7) als volgt:

$$\begin{cases} \dot{q} = Du + d_0 \\ M_\tau \dot{u} = h_\tau^* \end{cases} \tag{4.8}$$

De vergelijkingen van (4.8) zijn gelijk aan de initiële bewegingsvergelijkingen (2.1) voor een onbeperkt (unconstrained) systeem. De dimensie van de set van bepalende vergelijkingen is gereduceerd tot $2k$ en bestaat alleen uit de onafhankelijke snelheden en versnellingen.

T.g.v de bekende relatie (4.3) is het niet noodzakelijk om de kxn matrix C_λ als een orthogonaal complement van C_τ te berekenen, wat moeilijk kan zijn tijdens practisch gebruik. Bovendien is de analytische formulering voor de veronderstelde constraints niet noodzakelijk voor het verkrijgen van de **tangentiële dynamische vergelijkingen** (4.7).
Er geldt namelijk ook $f(x(q,t),t)=0$.

Als de constraintkrachten gevonden willen worden, moeten C_λ en $c_{\lambda 0}^*$ bepaald worden uit de relaties (2.4) en (2.5).
Na substitutie van $x=x(q,t)$ en $v=v(u,x,t)$, kunnen de onbepaalde multipliers en constraintkrachten gevonden worden uit de relaties (3.5) en (3.6) als functie van u, q en t.

A5. De Projectiemethode geassocieërd met niet-holonome constraints.

Veronderstel nu dat een systeem wordt voorgeschreven door m 1e-orde kinematische (of niet-holonome) constraints:

$$\phi(v,x,t)=0 \tag{5.1}$$

met $\phi=[\phi_1,\ldots,\phi_m]^T$ tenminste 1x differentieërbare functies, waarbij geeist wordt dat de rang van de Jacobi-matrix $\partial\phi/\partial v$ maximaal is. Vaak worden niet-holonome constraints in de volgende vorm geschreven:

$$\phi=Ev+e_0^*=0 \tag{5.2}$$

met $E(x,t)$ een mxn constraint matrix van maximale rang, en $e_0^*(x,t)$ een m-dimensionale vector. Vanwege de kinematische (gedifferentieërde) vorm van de geometrische constraints, wordt aangenomen dat (5.2) niet-integreerbaar is.
Analoog aan de holonome formulering in §6.4, mogen de niet-holonome constraintvgl. (5.1) of (5.2) geinterpreteerd worden als m nieuwe quasi-snelheden, die t.g.v. de opgelegde constraints, gelijk aan 0 blijven.
Dus $k(=n-m)$ onafhankelijke quasi-snelheden $u=[u_1,\ldots,u_k]^T$ kunnen geconstrueerd worden als volgt:

$$\begin{bmatrix}0\\u\end{bmatrix}=\begin{bmatrix}\phi(v,x,t)\\\gamma(v,x,t)\end{bmatrix}=\begin{bmatrix}Ev+e_0^*\\\gamma(v,x,t)\end{bmatrix} \tag{5.3}$$

Opgemerkt dient te worden dat de niet-holonomische constraints het aantal gegeneraliseerde coördinaten die de positie van het systeem in de n-dimensionale ruimte beschrijven **niet** reduceren.
Veronderstel nu, verwijzend naar (5.3) dat de rang van $((\partial\phi/\partial v)^T,(\partial\gamma/\partial v)^T)$=maximaal=$n$.
Dan bestaat er (tenminste theoretisch) een (inverse) relatie die er uitziet als $\boldsymbol{v=v(u,x,t)}$.

Bij practisch gebruik is het vaak moeilijk om deze relatie in eerste instantie te formuleren. Daarom wordt voor niet-lineaire niet-holonome constraints verwezen naar een algemene projectiemethode zoals ze beschreven staat in §6.3.

In het geval van lineaire niet-holonome constraints (5.2), kunnen de onafhankelijke quasi-snelheden gemakkelijk gedefineerd worden als (vergelijk met (3.9)):

$$M'\begin{bmatrix}0\\u\end{bmatrix}=\begin{bmatrix}0\\M_\tau u\end{bmatrix}=\begin{bmatrix}Ev+e_0^*\\C_\tau Mv\end{bmatrix}=TMv+\begin{bmatrix}e_0^*\\0\end{bmatrix} \tag{5.4}$$

met C_τ een orthogonaal complement van E, d.w.z. $C_\tau E^T=0$, en M_τ is gedefinieerd door relatie (3.7).
Differentiëren van (5.4) naar de tijd geeft

$$\begin{bmatrix}0\\M_\tau\dot{u}+\dot{M}_\tau u\end{bmatrix}=\begin{bmatrix}E\dot{v}+\dot{E}v+\dot{e}_0^*\\C_\tau M\dot{v}+(C_\tau M)^{\cdot}v\end{bmatrix}$$

en vervolgens herschrijven in de vorm van (3.9) met

$$\begin{aligned}
&C_\lambda=E && ..(a)\\
&c_{\lambda 0}^*=\dot{E}v+\dot{e}_0^* && ..(b)\\
&c_{\tau 0}^*=(C_\tau M)^{\cdot}v-\dot{M}_\tau u && ..(c)
\end{aligned} \tag{5.5}$$

Uit (5.4) volgt, mede door een beschouwing van orthogonaliteit, dat geldt

$$v=C_\tau^T u-M^{-1}C_\lambda^T M_\lambda^{-1}e_0^* \tag{5.6}$$

Nu kunnen de uiteindelijke bewegingsvergelijkingen als volgt uitgedrukt worden (vergelijk met (3.12)):

$$\begin{aligned}
&M_\tau(x,t)\,\dot{u}=h_\tau^*(u,x,t) && ..(a)\\
&\dot{x}=A(x,t)\,C_\tau^T(x,t)\,u+\tilde{a}_0(x,t) && ..(b)\\
&\quad ;\ \tilde{a}_0(x,t)=a_0-M^{-1}C_\lambda^T M_\lambda^{-1}e_0^*
\end{aligned} \tag{5.7}$$

De dimensie van (5.7) is gereduceerd tot $2n-m=k+n$ en de oplossing van (5.7) verzekert dat aan de constraints van (5.1) in principe is voldaan. Met de vergelijkingen (3.5) en (3.6) en rekening houdend met de nieuwe betekenis van de relaties, kunnen de responsies van de niet-holonomische constraints berekend worden als functie van de nieuwe waarden u, x en t.

APPENDIX AA

AA1. Kromlijnige coördinaten.

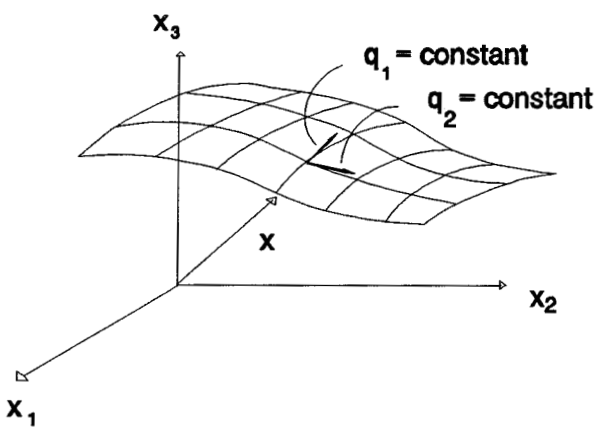
Bekijk een enkel deeltje gekarakteriseerd door 2 onafhankelijke gegeneraliseerde coördinaten q_1, q_2. De positievector van het deeltje t.o.v. een inertieël carthesisch coördinatenstelsel is $x=(x_1,x_2,x_3)^T$, met $x=x(q,t)$. Dan kan de snelheid van het deeltje uitgedrukt worden in de volgende relatie

$$\frac{dx}{dt}=J\dot{q}+\frac{\partial x}{\partial t} \tag{1}$$

met $\dot{q}=(\dot{q}_1,\dot{q}_2)$ en

$$J=\begin{bmatrix} \frac{\partial x_1}{\partial q_1} & \frac{\partial x_1}{\partial q_2} \\ \frac{\partial x_2}{\partial q_1} & \frac{\partial x_2}{\partial q_2} \\ \frac{\partial x_3}{\partial q_1} & \frac{\partial x_3}{\partial q_2} \end{bmatrix} \tag{2}$$

De transformatie tussen carthesische coördinaten en gegeneraliseerde coördinaten kan meetkundig geinterpreteerd worden als een oppervlak met q_1 en q_2 als kromlijnige coördinaten.(fig. 1)



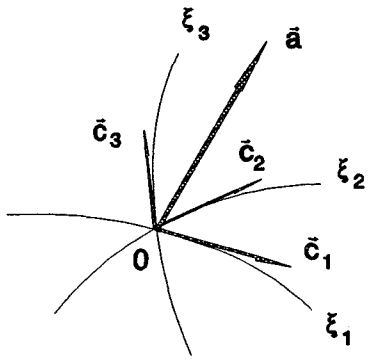
figuur 1. transformatie oppervlak

AA2. Covariant en Contravariant.

Stel nu een willekeurige basis (niet-orthonormaal)

$$\vec{c}_i = [\vec{c}_1 \ \vec{c}_2 \ \vec{c}_3] \tag{3}$$

Uitgangspunt zijn kromlijnige coördinaten waarbij de basis $\vec{c}_i$ wordt gedefinieerd als raaklijnen aan de kromlijninge coördinaten ξ_i (figuur 2).



figuur 2. Kromlijnige coördinaten

Dan kan een willekeurige vector $\vec{a}=[a_1 \ a_2 \ a_3]$ geschreven worden t.o.v. de willekeurige basis $\vec{c}_i$ als

$$\vec{a} = a_1\vec{c}_1 + a_2\vec{c}_2 + a_3\vec{c}_3 \tag{4}$$

waaarbij de componenten a_1, a_2 en a_3 de **contravariante componenten** zijn van deze willekeurige basis.

Als we nu vector $\vec{a}$ inwendig vermenigvuldingen met resp. $\vec{c}_1$, $\vec{c}_2$ en $\vec{c}_3$, krijgt men

$$\begin{bmatrix} \vec{a}\cdot\vec{c}_1 \\ \vec{a}\cdot\vec{c}_2 \\ \vec{a}\cdot\vec{c}_3 \end{bmatrix} = \begin{bmatrix} a_1\vec{c}_1\cdot\vec{c}_1 + a_2\vec{c}_2\cdot\vec{c}_1 + a_3\vec{c}_3\cdot\vec{c}_1 \\ a_1\vec{c}_1\cdot\vec{c}_2 + a_2\vec{c}_2\cdot\vec{c}_2 + a_3\vec{c}_3\cdot\vec{c}_2 \\ a_1\vec{c}_1\cdot\vec{c}_3 + a_2\vec{c}_2\cdot\vec{c}_3 + a_3\vec{c}_3\cdot\vec{c}_3 \end{bmatrix} = \begin{bmatrix} \vec{c}_1\cdot\vec{c}_1 + \vec{c}_2\cdot\vec{c}_1 + \vec{c}_3\cdot\vec{c}_1 \\ \vec{c}_1\cdot\vec{c}_2 + \vec{c}_2\cdot\vec{c}_2 + \vec{c}_3\cdot\vec{c}_2 \\ \vec{c}_1\cdot\vec{c}_3 + \vec{c}_2\cdot\vec{c}_3 + \vec{c}_3\cdot\vec{c}_3 \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \tag{5}$$

$$\begin{bmatrix} \vec{a}.\vec{c}_1 \\ \vec{a}.\vec{c}_2 \\ \vec{a}.\vec{c}_3 \end{bmatrix} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} \vec{c}_1 \\ \vec{c}_2 \\ \vec{c}_3 \end{bmatrix} . \vec{a} \qquad (6)$$

Nu zijn de scalairs α_1, α_2 en α_3 de **covariante componenten** van de willekeurige basis $\underline{c}_i$.
Verder geldt de relatie

$$\alpha_i = Ca_i \quad \Rightarrow \quad a_i = C^{-1}\alpha_i \qquad (7)$$

waarin de matric C^{-1} de **1e fundamentaalmatrix** wordt genoemd. Deze legt het verband tussen de covariante componenten en de contravariante componenten van de willekeurige basis $\underline{c}_i$.

Vergelijking (6) beschrijft het verband tussen de covariante componenten en de willekeurige vector $\underline{a}$.

De contravariante componenten kunnen op eenzelfde manier beschreven worden en wel als volgt:

$$a_i = C^{-1}\alpha_i = C^{-1} . \begin{bmatrix} \vec{c}_1 \\ \vec{c}_2 \\ \vec{c}_3 \end{bmatrix} . \vec{a} = \vec{\gamma}_i . \vec{a} \quad ; \quad \vec{\gamma}_i = \begin{bmatrix} \vec{\gamma}_1 \\ \vec{\gamma}_2 \\ \vec{\gamma}_3 \end{bmatrix} \qquad (8)$$

Dit betekent dat als $\vec{c}_i$ de basis is, $\vec{\gamma}_i$ een **reciproke basis** is van de willekeurige basis $\underline{c}_i$.

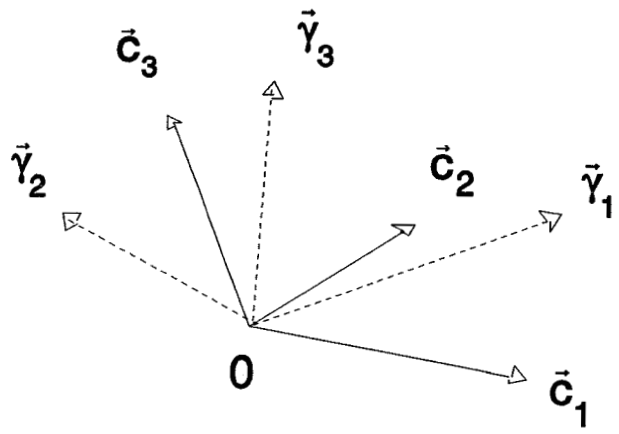
Dit houdt in dan de willekeurige vector $\vec{a}$ in termen van de reciproke basis geschreven kan worden als

$$\vec{a} = \alpha_1\vec{\gamma}_1 + \alpha_2\vec{\gamma}_2 + \alpha_3\vec{\gamma}_3 \qquad (9)$$

Vanuit (5) is af te leiden dat deze reciproke basis $\vec{\gamma}_i$ loodrecht op de willekeuringe basis $\vec{c}_i$ staat, zo dat geldt

$$\vec{\gamma}_i \vec{c}_i{}^T = I \Rightarrow \begin{cases} \vec{\gamma}_1 \cdot \vec{c}_1 = 1 & \vec{\gamma}_2 \cdot \vec{c}_1 = 0 & \vec{\gamma}_3 \cdot \vec{c}_1 = 0 \\ \vec{\gamma}_1 \cdot \vec{c}_2 = 0 & \vec{\gamma}_2 \cdot \vec{c}_2 = 1 & \vec{\gamma}_3 \cdot \vec{c}_2 = 0 \\ \vec{\gamma}_1 \cdot \vec{c}_3 = 0 & \vec{\gamma}_2 \cdot \vec{c}_3 = 0 & \vec{\gamma}_3 \cdot \vec{c}_3 = 1 \end{cases} \tag{10}$$

Dit kan grafisch duidelijk gemaakt worden in figuur 3.



figuur 3.

Deze figuur laat zien dat bij een bepaalde configuratie van de willekeuringe basis $\vec{c}_i$ er geldt:

$\vec{\gamma}_1$ loodrecht op $\vec{c}_2$, $\vec{c}_3$,

$\vec{\gamma}_2$ loodrecht op $\vec{c}_3$, $\vec{c}_1$, en

$\vec{\gamma}_3$ loodrecht op $\vec{c}_1$, $\vec{c}_2$.

AA3. Uitwerking van formule (4.4).

Uitgangspunt zijn resp. de formules (4.2), (3.4b) en (4.3) zoals ze hier gegeven zijn als:

$$\begin{aligned} x &= x(q,t) && ..(a) \\ \dot{x} &= Av + a_0 && ..(b) \\ \dot{q} &= Du + d_0 && ..(c) \end{aligned} \qquad (1)$$

Als we nu (1a) 1x differentieren naar de tijd, krijgt men

$$\dot{x} = \frac{\partial x}{\partial q}\frac{dq}{dt} + \frac{\partial x}{\partial t} = J\dot{q} + \frac{\partial x}{\partial t} \qquad (2)$$

Uit (1b) volgt

$$v = A^{-1}(\dot{x} - a_0) \qquad (3)$$

(1c) kan in (2) gesubstitueerd worden:

$$\dot{x} = JDu + Jd_0 + \frac{\partial x}{\partial t} \qquad (4)$$

Nu substitutie van (4) in (3):

$$\begin{aligned} v &= A^{-1}(JDu + Jd_0 + \frac{\partial x}{\partial t} - a_0) \\ &= A^{-1}JDu + A^{-1}(Jd_0 + \frac{\partial x}{\partial t} - a_0) \end{aligned} \qquad (5)$$

Om het resultaat van (4.4) te krijgen moet men (5) 1x differentieren naar de tijd. Er resulteert:

$$\begin{aligned} \dot{v} &= A^{-1}JD\dot{u} + (A^{-1}JD)\dot{u} + (A^{-1}[Jd_0 + \frac{\partial x}{\partial t} - a_0])^{\cdot} \\ &= A^{-1}JD\dot{u} + c_{u0} = C_\tau^T\dot{u} + c_{u0} \end{aligned} \qquad (4.4)$$

APPENDIX B

Coördinaten partitionering met LU-decompositie.

Bij de LU-decompositie wordt de Jacobiaan geveegd m.b.v Gauss-Jordan reductie, een veegoperatie die gebaseerd is op de methode van Gauss-Jordan [5].

Bij het pivotingproces wordt onderscheidt gemaakt tussen **partial pivoting** en **full pivoting.**
Partial pivoting wil zeggen dat met de grootste spil geveegd wordt die zich in de betreffende kolom bevindt waarin geveegd wordt.
Bij **full pivoting** wordt ook met de grootste spil geveegd, maar hier kijkt men naar alle kolommen en veegt men met die spil die de grootste waarde heeft.
Als de LU-decompositie met full pivoting wordt uitgevoerd op een mxn matrix A, kan dit resulteren in de volgende gepartioneerde vorm:

$$\begin{array}{c} \left[\begin{array}{c} \\ A \\ \\ \end{array}\right] m \\ n \end{array} \rightarrow \begin{array}{c} \left[\begin{array}{ccc:c} \ddots & U & & \\ & \ddots & & R \\ L & & \ddots & \\ \hdashline \cdots & \cdots & \cdots & \cdots \\ & S & & D \end{array}\right] \begin{array}{c} m-s \\ \\ \\ s \end{array} \\ \begin{array}{cc} m-s & n-m+s \end{array} \end{array}$$

Er wordt verondersteld dat er s overtollige rijen in de matrix zijn na decompositie t.g.v. full pivoting.
De rang van deze matrix is $m-s$. De L en U matrices nemen de $(m-s)x(m-s)$ linksboven elementen in, en D is een submatrix waarvan alle elementen beginnen op ongeveer nul(bv. kleiner dan een voorgeschreven tolerantie). De linkse $m-s$ kolommen van de bewerkte matrix worden de basiskolommen genoemd, en de overblijvende $n-m+s$ kolommen de niet-basiskolommen.

Als alle rijen van A onafhankelijk zijn, bv. als $s=0$, dan partitioneert de LU-decompositie A als volgt:

$$\begin{array}{c} \left[\begin{array}{c} \\ A \\ \\ \end{array}\right] m \\ n \end{array} \rightarrow \begin{array}{c} \left[\begin{array}{ccc:c} \ddots & U & & \\ & \ddots & & R \\ L & & \ddots & \end{array}\right] m \\ \begin{array}{cc} m & n-m \end{array} \end{array}$$

Hier representeert A de Jacobiaan $\Phi_{,q}$, waarbij geldt dat alle m constraints onafhankelijk zijn.

Omdat de elementen van q corresponderen met de kolomindices van $\Phi_{,q}$, definiëren de indices van de kolom van L (of U) de afhankelijke (basis) coördinaten u, en definiëren de indices van de kolommen van R de onafhankelijke (niet-basis) coördinaten v.

Partitionering van q in u en v correspondeerd met de partitionering van $\Phi_{,q}$ in $\Phi_{,u}$ en $\Phi_{,v}$. In termen van de matrices L, U en R wordt dit dan:

$$\Phi_{,u} = LU$$
$$\Phi_{,v} = LR$$

In sommige goed ontwikkelde LU-decompositie subroutines, wordt de matrix R vervangen door een matrix H, als volgt

$$\begin{bmatrix} \ddots & U & \vdots & \\ & \ddots & \vdots & R \\ L & \ddots & \vdots & \end{bmatrix} \rightarrow \begin{bmatrix} \ddots & U & \vdots & \\ & \ddots & \vdots & H \\ L & \ddots & \vdots & \end{bmatrix}$$

met

$$H = -U^{-1}R \quad \Rightarrow \quad UH = R$$

Hieruit volgt dat

$$\Phi_{,v} = -LUH = -\Phi_{,u}H \Rightarrow H = -\Phi_{,u}^{-1}\Phi_{,v}$$

Matrix H wordt de **influence coëfficienten matrix** genoemd. Deze matrix legt de relatie tussen variaties in u en variaties in v. Dit kan verkregen worden door de variatie in de constraint-vgl. $\Phi=0$ te nemen:

$$\delta\Phi = \Phi_{,q}\delta q = 0 \Rightarrow \Phi_{,u}\delta u + \Phi_{,v}\delta v = 0$$

Dit kan omgeschreven worden naar

$$\delta u = -\Phi_{,u}^{-1}\Phi_{,v}\delta v = H\delta v$$

De kinematische snelheidsvgl. houdt dus ook in dat geldt:

$$\dot{u} = H\dot{v}$$

APPENDIX C

Uitbreiding van de afleiding in §6.3.3:

Hier zal worden afgeleid dat de snelheid $\dot{z}_D$ en de versnelling $\ddot{z}_D$ gelijk aan nul zijn en dat de snelheid $\dot{z}_I$ alle snelheidsinformatie bevat die nodig is voor de beweging van het systeem.

Na substitutie van (6.3.9) in de 1e vgl. van (6.3.7) volgt dat

$$\Phi_{,q} V \dot{z} = 0 \tag{1}$$

Partionering van V en $\dot{z}$ resulteert in

$$\Phi_{,q} V_D \dot{z}_D + \Phi_{,q} V_I \dot{z}_I = 0 \tag{2}$$

Uit de orthogonaliteit van $\Phi_{,q}^T$ en V_I blijkt, dat de tweede term van (2) gelijk is aan nul, en wordt (2) gereduceerd tot

$$\Phi_{,q} V_D \dot{z}_D = 0 \tag{3}$$

In het artikel van Liang en Lance [13] wordt uitgelegd dat $\Phi_{,q} V_D$ regulier is. Hieruit volgt dus dat

$$\dot{z}_D = 0 \tag{4}$$

Vgl. (6.3.9) nogmaals differentieren naar de tijd levert

$$\ddot{q} = V\ddot{z} + \dot{V}\dot{z} \tag{5}$$

Substitutie van (5) en (6.3.9) in de 2e vgl. van (6.3.7) geeft

$$\Phi_{,q}(V\ddot{z} + \dot{V}\dot{z}) + (\Phi_{,q}\dot{q})_{,q} V\dot{z} = 0 \tag{6}$$

Na partitionering van V, $\dot{z}$ en $\ddot{z}$ resteert er

$$\Phi_{,q}(V_D\ddot{z}_D + V_I\ddot{z}_I + \dot{V}_D\dot{z}_D + \dot{V}_I\dot{z}_I) + (\Phi_{,q}\dot{q})_{,q}(V_D\dot{z}_D + V_I\dot{z}_I) = 0 \tag{7}$$

Uit (6.3.6) en (4) volgt dat

$$\Phi_{,q} V_D \ddot{z}_D + [\Phi_{,q} \dot{V}_I + (\Phi_{,q} \dot{q})_{,q} V_I] \dot{z}_I = 0 \tag{8}$$

M.b.t (6.3.6) kan de tijdsafgeleide van de matrix $\Phi_{,q} V_I$ geschreven worden als

$$\frac{\partial}{\partial t} (\Phi_{,q} V_I) = \Phi_{,q} \dot{V}_I + (\Phi_{,q} \dot{q})_{,q} V_I \tag{9}$$

Hierdoor gaat (8) over in

$$\Phi_{,q} V_D \ddot{z}_D + \frac{\partial}{\partial t} (\Phi_{,q} V_I) \dot{z}_I = 0 \tag{10}$$

Omdat $\Phi_{,q}^T$ en V_I orthogonaal zijn resteert

$$\Phi_{,q} V_D \ddot{z}_D = 0 \tag{11}$$

Omdat $\Phi_{,q} V_I$ regulier is resteert er uit (11)

$$\ddot{z}_D = 0 \tag{12}$$

Dus alle snelheidsinformatie wordt gegeven door de snelheid $\dot{z}_I$. (dit geldt tevens voor de versnelling)

APPENDIX D

Hier wordt de listing van het zelfgeschreven programma voor het algoritme van de projectiemethode gepresenteerd.
In somminge subroutines worden variabelen gebruikt die op het eerste oog niet vanzelfsprekend zijn. Om duidelijk te maken wat ze voorstellen zullen ze in het volgende kort uitgelegt worden:

SUBROUTINE CZIPP:

MS=$(V_I^T M V_I)$,MSINV=$(V_I^T M V_I)^{-1}$, GS=$(V_I^T M \dot{V}_I)$,
MSINVGSZIP=$(V_I^T M V_I)^{-1}(V_I^T M \dot{V}_I)\dot{z}_I$
MSINVVIT=$(V_I^T M V_I)^{-1} V_I^T$
MSIVNVITQU=$(V_I^T M V_I)^{-1} V_I^T Q^A$, etc.

SUBROUTINE CALQPP:

VIZIPP=$V_I \ddot{z}_I$, VIPZIP=$\dot{V}_I \dot{z}_I$

SUBROUTINE VORT:

Hier wordt het recursief schema van de projectiemethode geprogrammeerd, d.w.z

P2T=p_2^T
V1TP2=$v_1^T p_2$; met p_2 wordt bedoeld de 2e kolom van matrix P
V1V1TP2=$(v_1^T p_2) v_1$; hier wordt net andersom geprogrammeerd omdat men de regels van vermenigvuldigen van een matrix/getal met een vector moet beachten
A2=norm van $(p_2-(v_1^T p_2) v_1)$
Iedere andere term in deze subroutine is op eenzelfde manier weergegeven.

SUBROUTINE CALVIP:

Hier wordt de afgeleide van de matrix V_I berekend: $\dot{V}_I$ m.b.v het recursief schema behorende bij het Gram-Schmidt proces.

P2PT=$\dot{p}_2^T$,B2=$\dot{v}_1^T p_2+v_1^T \dot{p}_2$

V1PA2=$(v_1^T p_2)\dot{v}_1$; hier wordt ook weer andersom geprogrammeerd om een getal met een vector te vermenigvuldigen.
A2S=$\dot{p}_2-\dot{v}_1^T p_2+v_1^T \dot{p}_2-(v_1^T p_2)\dot{v}_1$

A2PS=$(v_1^T p_2)(\dot{v}_1^T p_2+v_1^T \dot{p}_2)$, A2P=$\dot{a}_2$=A2PT(teller)/A2PN(noemer)

A2PT=$\dot{p}_2^T p_2-(v_1^T p_2)(\dot{v}_1^T p_2+v_1^T \dot{p}_2)$,A2PN=$(p_2^T p_2-(v_1^T p_2)^2)^{3/2}$

V2P=$\dot{v}_2$; V2P(I) is de 2e kolom van de orthogonale matrix V,etc.

Iedere andere term in deze subroutine is op eenzelfde manier geprogrammeerd.

```
C
C             NUMERIEK ALGORITME VOOR DE PROJECTIEMETHODE
C             ------------------------------------------
C             ------------------------------------------
C
      PROGRAM MAIN
      IMPLICIT NONE
C
      INTEGER M,N,NB,NMINM,NV(2),NVP(2),NDV,I,J,IFAIL,II,INAME(10),
     %        FILEW1,NOUT
      DOUBLE PRECISION T0,TEND,Q,QP,TM,QU,MA,B,R,RP,Y(8),TOL,W(8,18),
     %                 RES,ZIP,BP,QPP,VIP,ZIPP,VI,FI,FQ,GAMMA
C
      COMMON/BLOCK1/M,N,NMINM
      COMMON/BLOCK2/T0,TEND
      COMMON/BLOCK3/FI(4)
      COMMON/BLOCK4/TM
      COMMON/BLOCK5/FQ(4,6)
      COMMON/BLOCK6/Q(6)
      COMMON/BLOCK7/QP(6)
      COMMON/BLOCK8/QPP(6)
      COMMON/BLOCK9/ZIP(2)
      COMMON/BLOCK10/ZIPP(2)
      COMMON/BLOCK11/QU(6),MA(6,6)
      COMMON/BLOCK12/RES(150,19),II
      COMMON/BLOCK13/VI(6,2)
      COMMON/BLOCK14/VIP(6,2)
      COMMON/BLOCK15/GAMMA(4,6)
      COMMON/BLOCK16/B(2,6)
      COMMON/BLOCK17/BP(2,6)
      COMMON/BLOCK18/R(2)
      COMMON/BLOCK19/RP(2)
C
C    zelfgeschreven subroutine FCN die wordt aangeroepen door de
C    integratieroutine D02CAF
C
      EXTERNAL FCN
C
C   invoer van parameters die afhankelijk zijn van de keuze van
C   het te analyseren systeem, met NB=aantal lichamen, M=aantal
C   constraints, N=aantal gegeneraliseerde coordinaten, N-M=aantal
C   vrijheidsgraden, NDV=aantal differentiaalvgl. nodig voor de
C   integratieroutine
C
      NB=2
      M=4
      N=3*NB
      NMINM=N-M
      NDV=4*NMINM
C
C   inlezen van de beginpositie, -snelheid, en andere systeemdata
C
C   begintijdstip en eindtijdstip
C
      WRITE(1,99999)
      READ(1,*) T0,TEND
      TM=T0
C
C   beginpositie Q
C
      DO 10 I=1,NB-1
         WRITE(1,99998) I,I,I
         READ(1,*) Q(3*I-2),Q(3*I-1),Q(3*I)
   10 CONTINUE
```

```
      DO 15 I=2,NB
         WRITE(1,99997) I,I,I
         READ(1,*) Q(3*I-2),Q(3*I-1),Q(3*I)
   15 CONTINUE
      WRITE(1,99996)
      DO 20 I=1,N
         WRITE(1,99995) Q(I)
   20 CONTINUE
C
C   definieren van de booleanmatrix B
C
      DO 24 I=1,NMINM
         DO 25 J=1,N
            B(I,J)=0.0D0
   25    CONTINUE
   24 CONTINUE
C
      WRITE(1,99994) NMINM
      READ(1,*) (NV(I), I=1,NMINM)
      DO 30 I=1,NMINM
         R(I)=Q(NV(I))
         B(I,NV(I))=1.0D0
   30 CONTINUE
C
C   beginsnelheid QP
C
      DO 40 I=1,NB-1
         WRITE(1,99993) I,I,I
         READ(1,*) QP(3*I-2),QP(3*I-1),QP(3*I)
   40 CONTINUE
C
      DO 45 I=2,NB
         WRITE(1,99992) I,I,I
         READ(1,*) QP(3*I-2),QP(3*I-1),QP(3*I)
   45 CONTINUE
C
      WRITE(1,99991)
      DO 50 I=1,N
         WRITE(1,99995) QP(I)
   50 CONTINUE
C
C   berekening van de afgeleide BP van de booleanmatrix B
C
      DO 54 I=1,NMINM
         DO 55 J=1,N
            BP(I,J)=0.0D0
   55    CONTINUE
   54 CONTINUE
C
      WRITE(1,99990) NMINM
      READ(1,*) (NVP(I),I=1,NMINM)
      DO 60 I=1,NMINM
C-----------------------------------------
C        RP(I)=QP(NVP(I))
C-----------------------------------------
         RP(I)=1.0D0
         BP(I,NVP(I))=1.0D0
   60 CONTINUE
C
C   gegeneraliseerde krachten QU
C
      DO 70 I=1,NB-1
         WRITE(1,99989) I,I,I
         READ(1,*) QU(3*I-2),QU(3*I-1),QU(3*I)
```

```
   70 CONTINUE
C
      DO 75 I=2,NB
         WRITE(1,99988) I,I,I
         READ(1,*) QU(3*I-2),QU(3*I-1),QU(3*I)
   75 CONTINUE
C
C   massamatrix MA
C
      DO 77 I=1,N
         DO 78 J=1,N
            MA(I,J)=0.0D0
   78    CONTINUE
   77 CONTINUE
C
      DO 80 I=1,NB-1
         WRITE(1,99987) I,I
         READ(1,*) MA((3*I-2),(3*I-2)),MA((3*I-1),(3*I-1))
         MA((3*I),(3*I))=MA((3*I-1),(3*I-1))
   80 CONTINUE
C
      DO 85 I=2,NB
         WRITE(1,99986) I,I
         READ(1,*) MA((3*I-2),(3*I-2)),MA((3*I-1),(3*I-1))
         MA((3*I),(3*I))=MA((3*I-1),(3*I-1))
   85 CONTINUE
C
C   Berekening van Q m.b.v Newton-Raphson iteratie
C
      CALL CALCQ
C
C   Berekening van QP m.b.v. Newton-Raphson iteratie
C
      CALL CALCQP
C
C   Berekening van de Jacobiaan FIQ
C
      CALL CALFIQ
C
C   Berekening van de orthogonale matrix V m.b.v het
C   Gram-Schmidt proces en partitionering van de matrix V
C   in de deelmatrix VI (VD is niet van belang)
C
      CALL VORT
C
C   De transformatie van de gegeneraliseerde coordinatenvector QP
C   naar de onafhankelijke snelheidsvector ZIP en de definitie van
C   de boolean matrix B
C
      CALL CALZIP
C
C   Berekening van de tijdsafgeleide van de Jacobiaan
C
      CALL CGAMMA
C
C   Berekening van de tijdsafgeleide van VI
C
      CALL CALVIP
C
C   Berekening van de n-m onafhankelijke versnellingscoordinaten
C   gebaseerd op de vgl. ZIPP=inv(MS)*GS*ZIP+inv(MS)*(VI)T*QU
C
      CALL CZIPP
C
```

```
C   Berekening van de gegeneraliseerde versnellingsvector QPP als
C   QPP=VI*ZIPP+VIP*ZIP
C
      CALL CALQPP
C
C   Inlezen van de beginvoorwaarden die nodig zijn voor de
C   integratie-routine D02CAF
C
      DO 90 I=1,N
         Y(I)=Q(I)
   90 CONTINUE
C
      DO 100 I=1,NMINM
         Y(I+6)=ZIP(I)
  100 CONTINUE
C
C   Starten van de integratie-routine D02CAF, een variabele orde
C   en variabele stapgrootte integratie-subroutine, waarbij gebruik
C   wordt gemaakt van de Adams-methode en waarbij geintegreerd wordt
C   over een bereik; deze integratie-subroutine roept een zelfge-
C   schreven integratieroutine FCN aan
C
      WRITE(1,99985)
      READ(1,*) TOL
      IFAIL=0
      CALL D02CAF(TM,TEND,NDV,Y,TOL,FCN,W,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99984)
      IF (TOL.LT.0.0D0) WRITE(1,99983)
C
C   Opbergen van de resultaten in array "res"
C
      NOUT=6
      WRITE(NOUT,99982)
      READ(5,99981) (INAME(I),I=1,10)
      FILEW1=31
      OPEN(UNIT=FILEW1, FILE=INAME, STATUS='OVERWRITE')
      WRITE(NOUT,99980)
      DO 110 I=1,II
         WRITE(NOUT,99979) RES(I,1)
         WRITE(NOUT,99978) RES(I,2),RES(I,8),RES(I,14)
         WRITE(NOUT,99978) RES(I,3),RES(I,9),RES(I,15)
         WRITE(NOUT,99978) RES(I,4),RES(I,10),RES(I,16)
         WRITE(NOUT,99978) RES(I,5),RES(I,11),RES(I,17)
         WRITE(NOUT,99978) RES(I,6),RES(I,12),RES(I,18)
         WRITE(NOUT,99978) RES(I,7),RES(I,13),RES(I,19)
  110 CONTINUE
      CLOSE(FILEW1)
      STOP
C
99999 FORMAT(/,1X,'Geef T0 en TEND')
99998 FORMAT(/,1X,'Geef PHI1(',I2,'),X1(',I2,'),Y1(',I2,')')
99997 FORMAT(/,1X,'Geef PHI2(',I2,'),X2(',I2,'),Y2(',I2,')')
99996 FORMAT(/,1X,'Q')
99995 FORMAT(6D12.4)
99994 FORMAT(/,1X,'Welke ',I2,' elementen van Q zijn accuraat?')
99993 FORMAT(/,1X,'Geef PHI1P(',I2,'),X1P(',I2,'),Y1P(',I2,')')
99992 FORMAT(/,1X,'Geef PHI2P(',I2,'),X2P(',I2,'),Y2P(',I2,')')
99991 FORMAT(/,1X,'QP')
99990 FORMAT(/,1X,'Welke ',I2,' elementen van QP zijn accuraat?')
99989 FORMAT(/,1X,'Geef QUPHI1(',I2,'),QUX1(',I2,'),QUY1(',I2,')')
99988 FORMAT(/,1X,'Geef QUPHI2(',I2,'),QUX2(',I2,'),QUY2(',I2,')')
99987 FORMAT(/,1X,'Geef J1(',I2,'),M1(',I2,')')
99986 FORMAT(/,1X,'Geef J2(',I2,'),M2(',I2,')')
99985 FORMAT(/,1X,'Geef de waarde van TOL')
```

```
99984 FORMAT('Error in D02CAF')
99983 FORMAT('Range too short for TOL')
99982 FORMAT(/,1X,'Geef de naam van de uitvoerfile')
99981 FORMAT(10A2)
99980 FORMAT('T',11X,'Q',11X,'QP',10X,'QPP')
99979 FORMAT(/,D12.4)
99978 FORMAT(18X,3D12.4)
C
      END
C
C ***************************************************************
C
      SUBROUTINE CALFIQ
C     -----------------
C de constraintmatrix FI en de Jacobiaan FQ moeten analystisch
C bepaald worden
C
      INTEGER I
      DOUBLE PRECISION L,Q,FI,FQ
C
      COMMON/BLOCK3/FI(4)
      COMMON/BLOCK5/FQ(4,6)
      COMMON/BLOCK6/Q(6)
C
      L=0.5D0
C
      FI(1)=Q(2)-0.5D0*L*SIN(Q(1))
      FI(2)=Q(3)-0.5D0*L*COS(Q(1))
      FI(3)=L*SIN(Q(1))-Q(5)+0.5D0*L*SIN(Q(4))
      FI(4)=L*COS(Q(1))-Q(6)+0.5D0*L*COS(Q(4))
C
      FQ(1,1)=-0.5D0*L*COS(Q(1))
      FQ(1,2)=1.0D0
C
      DO 200 I=3,6
         FQ(1,I)=0.0D0
  200 CONTINUE
C
      FQ(2,1)=0.5D0*L*SIN(Q(1))
      FQ(2,2)=0.0D0
      FQ(2,3)=1.0D0
C
      DO 210 I=4,6
         FQ(2,I)=0.0D0
  210 CONTINUE
C
      FQ(3,1)=L*COS(Q(1))
      FQ(3,2)=0.0D0
      FQ(3,3)=0.0D0
      FQ(3,4)=0.5D0*L*COS(Q(4))
      FQ(3,5)=-1.0D0
      FQ(3,6)=0.0D0
C
      FQ(4,1)=-L*SIN(Q(1))
      FQ(4,2)=0.0D0
      FQ(4,3)=0.0D0
      FQ(4,4)=-0.5D0*L*SIN(Q(4))
      FQ(4,5)=0.0D0
      FQ(4,6)=-1.0D0
C
      RETURN
      END
C
C ***************************************************************
```

```
C
      SUBROUTINE CALCQ
C     ----------------
C
      INTEGER N,M,NMINM,I,J,IFAIL
      DOUBLE PRECISION FI,Q,FQ,ACC,LENGDQ,A(6,6),B,C(6),DQ(6,1),
     %                 AA(6,6),WS2(6),WS3(6),BQ(2),R,WS1(1),DQT(1,6),
     %                 DQTDQ,WSDQ(1)
C
      COMMON/BLOCK1/M,N,NMINM
      COMMON/BLOCK3/FI(4)
      COMMON/BLOCK5/FQ(4,6)
      COMMON/BLOCK6/Q(6)
      COMMON/BLOCK16/B(2,6)
      COMMON/BLOCK18/R(2)
C
      ACC=0.1D-3
      LENGDQ=ACC
C
      DO 240 I=1,NMINM
         DO 230 J=1,N
            A(M+I,J)=B(I,J)
  230    CONTINUE
  240 CONTINUE
C
  250 IF (LENGDQ.LT.ACC) GOTO 310
C
      CALL CALFIQ
C
      DO 260 I=1,M
         DO 270 J=1,N
            A(I,J)=FQ(I,J)
  270    CONTINUE
  260 CONTINUE
C
      DO 280 I=1,M
         C(I)=-FI(I)
  280 CONTINUE
C
      IFAIL=0
      CALL F01CKF(BQ,B,Q,NMINM,1,N,WS1,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99953)
C
      DO 290 I=1,NMINM
         C(M+I)=R(I)-BQ(I)
  290 CONTINUE
C
      IFAIL=0
      CALL F04ATF(A,N,C,N,DQ,AA,N,WS2,WS3,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99952)
C
      DO 295 I=1,N
         DQT(1,I)=DQ(I,1)
  295 CONTINUE
C
      IFAIL=0
      CALL F01CKF(DQTDQ,DQT,DQ,1,1,N,WSDQ,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(99951)
C
      LENGDQ=SQRT(DQTDQ)
C
      DO 300 I=1,N
         Q(I)=DQ(I,1)+Q(I)
  300 CONTINUE
```

```
C
      GOTO 250
C
  310 RETURN
C
99953 FORMAT('Error in F01CKF:berekening van BQ')
99952 FORMAT('Error in F04ATF')
99951 FORMAT('Error in F01CKF:berekening van DQTDQ')
C
      END
C
C ****************************************************************
C
      SUBROUTINE CALCQP
C     -----------------
C
      INTEGER M,N,NMINM,I,J,IFAIL
      DOUBLE PRECISION FQ,B,RP,A(6,6),C(6),AA(6,6),WS2(6),WS3(6),QP,
     %                 VQP(6)
C
      COMMON/BLOCK1/M,N,NMINM
      COMMON/BLOCK5/FQ(4,6)
      COMMON/BLOCK7/QP(6)
      COMMON/BLOCK16/B(2,6)
      COMMON/BLOCK19/RP(2)
C
      DO 320 I=1,NMINM
         DO 330 J=1,N
            A(M+I,J)=B(I,J)
  330    CONTINUE
  320 CONTINUE
C
      CALL CALFIQ
C
      DO 340 I=1,M
         DO 350 J=1,N
            A(I,J)=FQ(I,J)
  350    CONTINUE
  340 CONTINUE
C
      DO 360 I=1,M
         C(I)=0.0D0
  360 CONTINUE
C
      DO 370 I=1,NMINM
         C(M+I)=RP(I)
  370 CONTINUE
C
      IFAIL=0
      CALL F04ATF(A,N,C,N,VQP,AA,N,WS2,WS3,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99949)
C
      DO 375 I=1,N
         QP(I)=VQP(I)
  375 CONTINUE
C
      RETURN
C
99949 FORMAT('Error in F04ATF')
C
      END
C
C ****************************************************************
C
```

```
      SUBROUTINE CALZIP
C     -----------------
C
      INTEGER N,NMINM,I,J,IFAIL,M
      DOUBLE PRECISION ZIP,VI,QP,MQP(6),MZIP(2),WSB(1),B,VIT(2,6),
     %                 MVI(6,2)
C
      COMMON/BLOCK1/M,N,NMINM
      COMMON/BLOCK7/QP(6)
      COMMON/BLOCK10/ZIP(2)
      COMMON/BLOCK13/VI(6,2)
      COMMON/BLOCK16/B(2,6)
C
      DO 380 I=1,N
         MQP(I)=QP(I)
  380 CONTINUE
C
      CALL VORT
C
      DO 385 I=1,N
         DO 386 J=1,NMINM
            MVI(I,J)=VI(I,J)
            VIT(J,I)=MVI(I,J)
  386    CONTINUE
  385 CONTINUE
C
      IFAIL=0
      CALL F01CKF(MZIP,VIT,MQP,NMINM,1,N,WSB,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99947)
C
      DO 390 I=1,NMINM
         ZIP(I)=MZIP(I)
  390 CONTINUE
C
C matrix B wordt hergedefinieerd als B=(VI)T
C
      DO 400 I=1,NMINM
         DO 410 J=1,N
            B(I,J)=VIT(I,J)
  410    CONTINUE
  400 CONTINUE
C
      RETURN
C
99947 FORMAT('Error in F01CKF;berekening van ZIP')
C
      END
C
C ****************************************************************
C
      SUBROUTINE CGAMMA
C     -----------------
C de matrix gamma=(phi,q*qp),q moet analytisch bepaald worden
C
      INTEGER I
      DOUBLE PRECISION Q,QP,GAMMA,L
C
      COMMON/BLOCK6/Q(6)
      COMMON/BLOCK7/QP(6)
      COMMON/BLOCK15/GAMMA(4,6)
C
      L=0.5D0
C
      GAMMA(1,1)=0.5D0*L*QP(1)*SIN(Q(1))
```

```
C
      DO 411 I=2,6
         GAMMA(1,I)=0.0D0
  411 CONTINUE
C
      GAMMA(2,1)=0.5D0*L*QP(1)*COS(Q(1))
C
      DO 412 I=2,6
         GAMMA(2,I)=0.0D0
  412 CONTINUE
C
      GAMMA(3,1)=-L*QP(1)*SIN(Q(1))
      GAMMA(3,2)=0.0D0
      GAMMA(3,3)=0.0D0
      GAMMA(3,4)=-0.5D0*L*QP(4)*SIN(Q(4))
      GAMMA(3,5)=0.0D0
      GAMMA(3,6)=0.0D0
C
      GAMMA(4,1)=-L*QP(1)*COS(Q(1))
      GAMMA(4,2)=0.0D0
      GAMMA(4,3)=0.0D0
      GAMMA(4,4)=-0.5D0*L*(QP(4)**2)*COS(Q(4))
      GAMMA(4,5)=0.0D0
      GAMMA(4,6)=0.0D0
C
      RETURN
C
      END
C
C ****************************************************************
C
      SUBROUTINE CZIPP
C     ----------------
C berekening van de onafhankelijke versnellingen zipp als
C ZIPP=inv(MS)*GS*ZIP+inv(MS)*(VI)T*QU
C
      INTEGER N,NMINM,I,J,IFAIL,M
      DOUBLE PRECISION ZIP,QU,VI,VIP,MA,MAVI(6,2),AD,BC,COEF,
     %                 MS(2,2),MSINV(2,2),GS(2,2),WS1(1),
     %                 WS5(1),WS7(1),WS8(1),WS2(1),MSINVGS(2,2),
     %                 MSINVGSZIP(2),MSINVVIT(2,6),ZIPP,WS9(1),
     %                 WS10(1),MSINVVITQU(2),MAVIP(6,2),VZIP(2),
     %                 MVI(6,2),WS3(1),VMSINVGSZIP(2),VMSINVVIT(2,6),
     %                 VMSINVVITQU(2),MMAVI(6,2),VIT(2,6),MMS(2,2),
     %                 MMAVIP(6,2),MMSINVGS(2,2),MGS(2,2)
C
      COMMON/BLOCK1/M,N,NMINM
      COMMON/BLOCK9/ZIP(2)
      COMMON/BLOCK10/ZIPP(2)
      COMMON/BLOCK11/QU(6),MA(6,6)
      COMMON/BLOCK13/VI(6,2)
      COMMON/BLOCK14/VIP(6,2)
C
      CALL VORT
C
C berekening van de term MS=inv(vi,t*m*vi) van vgl. (31)
C
      DO 413 I=1,N
         DO 414 J=1,NMINM
            MVI(I,J)=VI(I,J)
  414    CONTINUE
  413 CONTINUE
C
      IFAIL=0
```

```
      CALL F01CKF(MMAVI,MA,MVI,N,NMINM,N,WS1,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99946)
C
      DO 415 I=1,N
         DO 416 J=1,NMINM
            MAVI(I,J)=MMAVI(I,J)
            VIT(J,I)=VI(I,J)
  416    CONTINUE
  415 CONTINUE
C
      IFAIL=0
      CALL F01CKF(MMS,VIT,MAVI,NMINM,NMINM,N,WS2,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99945)
C
      DO 417 I=1,NMINM
         DO 418 J=1,NMINM
            MS(I,J)=MMS(I,J)
            MSINV(I,J)=0.0D0
  418    CONTINUE
  417 CONTINUE
C
      AD=MS(1,1)*MS(2,2)
      BC=MS(1,2)*MS(2,1)
      COEF=1/(AD-BC)
      MSINV(1,1)=COEF*MS(2,2)
      MSINV(1,2)=-COEF*MS(1,2)
      MSINV(2,1)=-COEF*MS(2,1)
      MSINV(2,2)=COEF*MS(1,1)
C-----------------------------------------------------------------
C     IFAIL=0
C     CALL F01AAF(MS,NMINM,2,MMSINV,2,WS4,IFAIL)
C     IF (IFAIL.GT.1) WRITE(1,99944)
C
C     DO 419 I=1,NMINM
C        DO 420 J=1,NMINM
C           MSINV(I,J)=MMSINV(I,J)
C 420    CONTINUE
C 419 CONTINUE
C-----------------------------------------------------------------
C berekening van de term GS=vi,t*m*vipunt van vgl. (31)
C
      CALL CALVIP
C
      IFAIL=0
      CALL F01CKF(MMAVIP,MA,VIP,N,NMINM,N,WS5,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99943)
C
      DO 421 I=N
         DO 422 J=1,NMINM
            MAVIP(I,J)=MMAVIP(I,J)
  422    CONTINUE
  421 CONTINUE
C
      IFAIL=0
      CALL F01CKF(MGS,VIT,MAVIP,NMINM,NMINM,N,WS3,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99942)
C
      DO 423 I=1,NMINM
         DO 424 J=1,NMINM
            GS(I,J)=MGS(I,J)
  424    CONTINUE
  423 CONTINUE
C
C berekening van de term inv(MS)*GS
```

```
C
      IFAIL=0
      CALL F01CKF(MMSINVGS,MSINV,GS,NMINM,NMINM,NMINM,WS7,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99941)
C
      DO 425 I=1,NMINM
         DO 426 J=1,NMINM
            MSINVGS(I,J)=MMSINVGS(I,J)
  426    CONTINUE
  425 CONTINUE
C
      CALL CALZIP
C
      DO 427 I=1,NMINM
         VZIP(I)=ZIP(I)
  427 CONTINUE
C
C berekening van de eerste term in vgl. (31)
C
      IFAIL=0
      CALL F01CKF(VMSINVGSZIP,MSINVGS,VZIP,NMINM,1,NMINM,WS8,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99940)
C
      DO 430 I=1,NMINM
         MSINVGSZIP(I)=VMSINVGSZIP(I)
  430 CONTINUE
C
C berekening van de term inv(MS)*(vi)T
C
      IFAIL=0
      CALL F01CKF(VMSINVVIT,MSINV,VIT,NMINM,N,NMINM,WS9,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99939)
C
      DO 440 I=1,NMINM
         DO 450 J=1,N
            MSINVVIT(I,J)=VMSINVVIT(I,J)
  450    CONTINUE
  440 CONTINUE
C
C berekening van de tweede term in vgl. (31)
C
      IFAIL=0
      CALL F01CKF(VMSINVVITQU,MSINVVIT,QU,NMINM,1,N,WS10,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99938)
C
      DO 460 I=1,NMINM
         MSINVVITQU(I)=VMSINVVITQU(I)
  460 CONTINUE
C
      DO 470 I=1,NMINM
         ZIPP(I)=MSINVGSZIP(I)+MSINVVITQU(I)
  470 CONTINUE
C
      RETURN
C
99946 FORMAT('Error in F01CKF;berekening van MAVI')
99945 FORMAT('Error in F01CKF;berekening van MS')
99944 FORMAT('Error in F01AAF')
99943 FORMAT('Error in F01CKF;berekening van MAVIP')
99942 FORMAT('Error in F01CKF;berekening van GS')
99941 FORMAT('Error in F01CKF;berekening van MSINVGS')
99940 FORMAT('Error in F01CKF;berekening van MSINVGSZIP')
99939 FORMAT('Error in F01CKF;berekening van MSINVVIT')
99938 FORMAT('Error in F01CKF;berekening van MSINVVITQU')
```

```
C
      END
C
C ***************************************************************
C
      SUBROUTINE CALQPP
C     -----------------
C
      INTEGER N,NMINM,I,J,IFAIL,M
      DOUBLE PRECISION ZIPP,VZIPP(2),VI,MVI(6,2),QPP,
     %                 VIZIPP(6),MVIZIPP(6),VIPZIP(6),MVIPZIP(6),
     %                 VZIP(2),ZIP,MVIP(6,2),VIP,WSD(1),WSC(1)
C
      COMMON/BLOCK1/M,N,NMINM
      COMMON/BLOCK8/QPP(6)
      COMMON/BLOCK9/ZIP(2)
      COMMON/BLOCK10/ZIPP(2)
      COMMON/BLOCK13/VI(6,2)
      COMMON/BLOCK14/VIP(6,2)
C
      CALL CZIPP
C
      DO 480 I=1,NMINM
         VZIPP(I)=ZIPP(I)
  480 CONTINUE
C
      CALL VORT
C
      DO 490 I=1,N
         DO 500 J=1,NMINM
            MVI(I,J)=VI(I,J)
  500    CONTINUE
  490 CONTINUE
C
      IFAIL=0
      CALL F01CKF(MVIZIPP,MVI,VZIPP,N,1,NMINM,WSC,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99934)
C
      DO 510 I=1,N
         VIZIPP(I)=MVIZIPP(I)
  510 CONTINUE
C
      CALL CALVIP
C
      DO 520 I=1,N
         DO 530 J=1,NMINM
            MVIP(I,J)=VIP(I,J)
  530    CONTINUE
  520 CONTINUE
C
      CALL CALZIP
C
      DO 540 I=1,NMINM
         VZIP(I)=ZIP(I)
  540 CONTINUE
C
      IFAIL=0
      CALL F01CKF(MVIPZIP,MVIP,VZIP,N,1,NMINM,WSD,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99933)
C
      DO 550 I=1,N
         VIPZIP(I)=MVIPZIP(I)
  550 CONTINUE
C
```

```
      DO 560 I=1,N
         QPP(I)=VIZIPP(I)+VIPZIP(I)
  560 CONTINUE
C
      RETURN
C
99934 FORMAT('Error in F01CKF;berekening van MVIZIPP')
99933 FORMAT('Error in F01CKF;berekening van MVIPZIP')
C
      END
C
C ****************************************************************
C
      SUBROUTINE CALQPI
C     -----------------
C hier wordt qp(i+1) berekent na iedere integratiestap, dus nadat
C qpp(i+1) bekent is en de boolean matrix B hergedefinieert is als
C B=(VI)T(i+1)
C
      INTEGER I,J,N,NMINM,IFAIL,M
      DOUBLE PRECISION VI,MVI(6,2),ZIP,VZIP(2),QP,VQP(6),WSE(1)
C
      COMMON/BLOCK1/M,N,NMINM
      COMMON/BLOCK7/QP(6)
      COMMON/BLOCK9/ZIP(2)
      COMMON/BLOCK13/VI(6,2)
C
      CALL CALZIP
C
      DO 570 I=1,NMINM
         VZIP(I)=ZIP(I)
  570 CONTINUE
C
      CALL VORT
C
      DO 580 I=1,N
         DO 590 J=1,NMINM
            MVI(I,J)=VI(I,J)
  590    CONTINUE
  580 CONTINUE
C
      IFAIL=0
      CALL F01CKF(VQP,MVI,VZIP,N,1,NMINM,WSE,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99932)
C
      DO 600 I=1,N
         QP(I)=VQP(I)
  600 CONTINUE
C
      RETURN
C
99932 FORMAT('Error in F01CKF:berekening van VQPI')
C
      END
C
C ****************************************************************
C
      SUBROUTINE FCN(T,Y,F)
C     ---------------------
C
      INTEGER M,N,NMINM,I,II
      DOUBLE PRECISION TEND,Q,ZIP,Y(8),F(8),STEP,VI,VIP,ZIPP,QPP,RES,
     %                 QP,FI,FQ,QU,MA,GAMMA,T0,B,BP,R,RP,T,TM
C
```

```
      COMMON/BLOCK1/M,N,NMINM
      COMMON/BLOCK2/T0,TEND
      COMMON/BLOCK3/FI(4)
      COMMON/BLOCK4/TM
      COMMON/BLOCK5/FQ(4,6)
      COMMON/BLOCK6/Q(6)
      COMMON/BLOCK7/QP(6)
      COMMON/BLOCK8/QPP(6)
      COMMON/BLOCK9/ZIP(2)
      COMMON/BLOCK10/ZIPP(2)
      COMMON/BLOCK11/QU(6),MA(6,6)
      COMMON/BLOCK12/RES(150,19),II
      COMMON/BLOCK13/VI(6,2)
      COMMON/BLOCK14/VIP(6,2)
      COMMON/BLOCK15/GAMMA(4,6)
      COMMON/BLOCK16/B(2,6)
      COMMON/BLOCK17/BP(2,6)
      COMMON/BLOCK18/R(2)
      COMMON/BLOCK19/RP(2)
C
      TM=T
      WRITE(1,99931) TM
C
C a: berekening van Q en ZIP
C-----------------------------------------------------------------
C     WRITE(1,99930)
C     DO 610 I=1,N
C        Q(I)=Y(I)
C        DO 620 J=1,NMINM
C           ZIP(J)=Y(J+6)
C     WRITE(1,99930) Q(I),ZIP(J)
C 620 CONTINUE
C 610 CONTINUE
C-----------------------------------------------------------------
      WRITE(1,99930)
      DO 610 I=1,N
         Q(I)=Y(I)
         WRITE(1,99929) Q(I)
  610 CONTINUE
C
      WRITE(1,99924)
      DO 620 I=1,NMINM
         ZIP(I)=Y(N+I)
      WRITE(1,99929) ZIP(I)
  620 CONTINUE
C
      IF (II.EQ.0) GOTO 640
C
C b: berekening van (VI(i+1))T uit het Gram-Schmidt proces
C
      CALL VORT
C
C c: berekening van qp(i+1)
C
      CALL CALQPI
C
C d: berekening van de tijdsafgeleide van de Jacobiaan en VI(i+1)
C
      CAll CGAMMA
C
      CALL CALVIP
C
C e: berekening van de onafhankelijke versnellingen
C
```

```
      CALL CZIPP
C
C f: berekening van de gegeneraliseerde versnellingen
C
      CALL CALQPP
C
      GOTO 650
C
  640 WRITE(1,99928)
      READ(1,*) STEP
C
C g: resultaten uitvoeren naar het scherm
C
  650 WRITE(1,99927)
      DO 660 I=1,N
         WRITE(1,99926) Q(I),QP(I),QPP(I)
  660 CONTINUE
      WRITE(1,99925)
      DO 670 I=1,NMINM
         WRITE(1,99926) ZIP(I),ZIPP(I)
  670 CONTINUE
C
C begincondities voor de integratie
C
      DO 680 I=1,N
         Y(I)=Q(I)
         F(I)=QP(I)
  680 CONTINUE
C
      DO 690 I=1,NMINM
         Y(I+6)=ZIP(I)
         F(I+6)=ZIPP(I)
  690 CONTINUE
C
C opbergen van de resultaten in array "res"
C
  692 IF (II.EQ.0) GOTO 694
      IF (TM.GT.RES(II,1)+STEP) GOTO 694
      IF (TM.GT.RES(II,1)) GOTO 698
      II=II-1
      GOTO 692
  694 II=II+1
      RES(II,1)=TM
      DO 696 I=1,6
         RES(II,1+I)=Q(I)
         RES(II,7+I)=QP(I)
         RES(II,13+I)=QPP(I)
  696 CONTINUE
      GOTO 699
  698 IF (TM.EQ.TEND) GOTO 694
  699 RETURN
C
99931 FORMAT(/,1X,1X,'T= ',D12.4,' SEC')
99930 FORMAT(/,1X,' Q')
99929 FORMAT(2D12.4)
99928 FORMAT(/,1X,'Geef de waarde van STEP')
99927 FORMAT(/,1X,' Q',11X,'QP',10X,'QPP')
99926 FORMAT(3D12.4)
99925 FORMAT(/,1X,'ZIP',9X,'ZIPP')
99924 FORMAT(/,1X,'ZIP')
C
      END
C
C ******************************************************************
```

```
C
      SUBROUTINE VORT
C     ---------------
C
      INTEGER N,M,NMINM,I,J,IFAIL
      DOUBLE PRECISION P(6,6),V(6,6),BT(6,2),FQT(6,4),WSH(1),WSBB(1),
     %                 FQ,B,WSJ(1),WSM(1),WSN(1),WSQ(1),WST(1),WSU(1),
     %                 WSW(1),WSNA(1),WSCA(1),WSDA(1),WSFA(1),WSHA(1),
     %                 WSKA(1),WSOA(1),WSZ(1),WSQA(1),WSSA(1),WSUA(1),
     %                 WSXA(1),A6,AA6,P1T(1,6),P1(6),V1(6),V1T(1,6),
     %                 P2(6),A22(6,1),A22T(1,6),V2(6),V2T(1,6),P3(6),
     %                 V1V1TP3(6),V2V2TP3(6),A33(6,1),A33T(1,6),V3(6),
     %                 V3T(1,6),P4(6),V1V1TP4(6),V2V2TP4(6),V3V3TP4(6),
     %                 A44(6,1),A44T(1,6),V4(6),V4T(1,6),P5(6),V5TP6,
     %                 V1V1TP5(6),V2V2TP5(6),V3V3TP5(6),V4V4TP5(6),
     %                 A55(6,1),A55T(1,6),V5(6),V5T(1,6),P6(6),V3TP5,
     %                 V1V1TP6(6),V2V2TP6(6),V3V3TP6(6),V4V4TP6(6),
     %                 V5V5TP6(6),A66(6,1),A66T(1,6),V6(6),VI,V4TP6,
     %                 V1V1TP2(6),AA1,A1,V1TP2,AA2,A2,V1TP3,V2TP3,AA3,
     %                 A3,V1TP4,V2TP4,V3TP4,AA4,A4,V1TP5,V2TP5,V4TP5,
     %                 AA5,A5,V1TP6,V2TP6,V3TP6
C
      COMMON/BLOCK1/M,N,NMINM
      COMMON/BLOCK5/FQ(4,6)
      COMMON/BLOCK13/VI(6,2)
      COMMON/BLOCK16/B(2,6)
C
      DO 700 I=1,N
         DO 710 J=1,NMINM
            BT(I,J)=B(J,I)
            P(I,M+J)=BT(I,J)
  710    CONTINUE
  700 CONTINUE
C
      CALL CALFIQ
C
      DO 720 I=1,N
         DO 730 J=1,M
            FQT(I,J)=FQ(J,I)
            P(I,J)=FQT(I,J)
  730    CONTINUE
  720 CONTINUE
C
C toepassen van het recursief Gram-Schmidt algoritme, beschreven in
C paragraaf 6.2.3, waarbij de stappen i=1,..,n geprogrammeerd worden
C
C ---------------------------stap 1: i=1---------------------------
C
      DO 740 I=1,N
         P1T(1,I)=P(I,1)
         P1(I)=P(I,1)
  740 CONTINUE
C
      IFAIL=0
      CALL F01CKF(AA1,P1T,P1,1,1,N,WSH,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99920)
C
      A1=1/(SQRT(AA1))
      DO 750 I=1,N
         V1(I)=P1(I)*A1
  750 CONTINUE
C
      DO 760 I=1,N
         V(I,1)=V1(I)
```

```
         V1T(1,I)=V(I,1)
  760 CONTINUE
C ---------------------------stap 2: i=2---------------------------
      DO 770 I=1,N
         P2(I)=P(I,2)
  770 CONTINUE
C
      IFAIL=0
      CALL F01CKF(V1TP2,V1T,P2,1,1,N,WSJ,1,1,IFAIL)
      IF (IFAIL.GT.O) WRITE(1,99919)
C
      DO 790 I=1,N
         V1V1TP2(I)=V1(I)*V1TP2
  790 CONTINUE
C
      DO 795 I=1,N
         A22(I,1)=P2(I)-V1V1TP2(I)
         A22T(1,I)=A22(I,1)
  795 CONTINUE
C
      IFAIL=0
      CALL F01CKF(AA2,A22T,A22,1,1,N,WSM,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99918)
C
      A2=1/(SQRT(AA2))
      DO 800 I=1,N
         V2(I)=A22(I,1)*A2
         V(I,2)=V2(I)
         V2T(1,I)=V(I,2)
  800 CONTINUE
C ---------------------------stap 3: i=3---------------------------
      DO 810 I=1,N
         P3(I)=P(I,3)
  810 CONTINUE
C
      IFAIL=0
      CALL F01CKF(V1TP3,V1T,P3,1,1,N,WSN,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99917)
C
      DO 830 I=1,N
         V1V1TP3(I)=V1(I)*V1TP3
  830 CONTINUE
C
      IFAIL=0
      CALL F01CKF(V2TP3,V2T,P3,1,1,N,WSQ,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99916)
C
      DO 850 I=1,N
         V2V2TP3(I)=V2(I)*V2TP3
  850 CONTINUE
C
      DO 855 I=1,N
         A33(I,1)=P3(I)-V1V1TP3(I)-V2V2TP3(I)
         A33T(1,I)=A33(I,1)
  855 CONTINUE
C
      IFAIL=0
      CALL F01CKF(AA3,A33T,A33,1,1,N,WST,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99915)
C
      A3=1/(SQRT(AA3))
      DO 860 I=1,N
         V3(I)=A33(I,1)*A3
         V(I,3)=V3(I)
```

```
  860 CONTINUE
C ---------------------------stap 4: i=4---------------------------
      DO 870 I=1,N
         P4(I)=P(I,4)
  870 CONTINUE
C
      IFAIL=0
      CALL F01CKF(V1TP4,V1T,P4,1,1,N,WSU,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99914)
C
      DO 890 I=1,N
         V1V1TP4(I)=V1(I)*V1TP4
  890 CONTINUE
C
      IFAIL=0
      CALL F01CKF(V2TP4,V2T,P4,1,1,N,WSW,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99913)
C
      DO 910 I=1,N
         V2V2TP4(I)=V2(I)*V2TP4
  910 CONTINUE
C
      DO 920 I=1,N
         V3T(1,I)=V(I,3)
  920 CONTINUE
C
      IFAIL=0
      CALL F01CKF(V3TP4,V3T,P4,1,1,N,WSZ,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99912)
C
      DO 930 I=1,N
         V3V3TP4(I)=V3(I)*V3TP4
  930 CONTINUE
C
      DO 935 I=1,N
         A44(I,1)=P4(I)-V1V1TP4(I)-V2V2TP4(I)-V3V3TP4(I)
         A44T(1,I)=A44(I,1)
  935 CONTINUE
C
      IFAIL=0
      CALL F01CKF(AA4,A44T,A44,1,1,N,WSCA,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99911)
C
      A4=1/(SQRT(AA4))
      DO 940 I=1,N
         V4(I)=A44(I,1)*A4
         V(I,4)=V4(I)
  940 CONTINUE
C ---------------------------stap 5: i=5---------------------------
      DO 950 I=1,N
         P5(I)=P(I,5)
  950 CONTINUE
C
      IFAIL=0
      CALL F01CKF(V1TP5,V1T,P5,1,1,N,WSDA,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99910)
C
      DO 970 I=1,N
         V1V1TP5(I)=V1(I)*V1TP5
  970 CONTINUE
C
      IFAIL=0
      CALL F01CKF(V2TP5,V2T,P5,1,1,N,WSFA,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99909)
```

```
C
      DO 990 I=1,N
         V2V2TP5(I)=V2(I)*V2TP5
  990 CONTINUE
C
      IFAIL=0
      CALL F01CKF(V3TP5,V3T,P5,1,1,N,WSHA,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99908)
C
      DO 1010 I=1,N
         V3V3TP5(I)=V3(I)*V3TP5
 1010 CONTINUE
C
      DO 1020 I=1,N
         V4T(1,I)=V(I,4)
 1020 CONTINUE
C
      IFAIL=0
      CALL F01CKF(V4TP5,V4T,P5,1,1,N,WSKA,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99907)
C
      DO 1030 I=1,N
         V4V4TP5(I)=V4(I)*V4TP5
 1030 CONTINUE
C
      DO 1035 I=1,N
         A55(I,1)=P5(I)-V1V1TP5(I)-V2V2TP5(I)-V3V3TP5(I)-V4V4TP5(I)
         A55T(1,I)=A55(I,1)
 1035 CONTINUE
C
      IFAIL=0
      CALL F01CKF(AA5,A55T,A55,1,1,N,WSNA,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99906)
C
      A5=1/(SQRT(AA5))
      DO 1040 I=1,N
         V5(I)=A55(I,1)*A5
         V(I,5)=V5(I)
 1040 CONTINUE
C ---------------------------stap 6: i=n=6------------------------
      DO 1050 I=1,N
         P6(I)=P(I,6)
 1050 CONTINUE
C
      IFAIL=0
      CALL F01CKF(V1TP6,V1T,P6,1,1,N,WSOA,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99905)
C
      DO 1070 I=1,N
         V1V1TP6(I)=V1(I)*V1TP6
 1070 CONTINUE
C
      IFAIL=0
      CALL F01CKF(V2TP6,V2T,P6,1,1,N,WSQA,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99904)
C
      DO 1090 I=1,N
         V2V2TP6(I)=V2(I)*V2TP6
 1090 CONTINUE
C
      IFAIL=0
      CALL F01CKF(V3TP6,V3T,P6,1,1,N,WSSA,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99903)
C
```

```
      DO 1110 I=1,N
         V3V3TP6(I)=V3(I)*V3TP6
 1110 CONTINUE
C
      IFAIL=0
      CALL F01CKF(V4TP6,V4T,P6,1,1,N,WSUA,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99902)
C
      DO 1130 I=1,N
         V4V4TP6(I)=V4(I)*V4TP6
 1130 CONTINUE
C
      DO 1140 I=1,N
         V5T(1,I)=V(I,5)
 1140 CONTINUE
C
      IFAIL=0
      CALL F01CKF(V5TP6,V5T,P6,1,1,N,WSXA,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99901)
C
      DO 1150 I=1,N
         V5V5TP6(I)=V5(I)*V5TP6
 1150 CONTINUE
C
      DO 1155 I=1,N
         A66(I,1)=P6(I)-V1V1TP6(I)-V2V2TP6(I)-V3V3TP6(I)-V4V4TP6(I)-
     %            V5V5TP6(I)
         A66T(1,I)=A66(I,1)
 1155 CONTINUE
C
      IFAIL=0
      CALL F01CKF(AA6,A66T,A66,1,1,N,WSBB,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99900)
C
      A6=1/(SQRT(AA6))
      DO 1160 I=1,N
         V6(I)=A66(I,1)*A6
         V(I,6)=V6(I)
 1160 CONTINUE
C
      DO 1170 I=1,N
         DO 1180 J=1,NMINM
            VI(I,J)=V(I,M+J)
 1180    CONTINUE
 1170 CONTINUE
C
      RETURN
C
99920 FORMAT('Error in F01CKF:berekening van A11')
99919 FORMAT('Error in F01CKF:berekening van V1TP2')
99918 FORMAT('Error in F01CKF:berekening van AA2')
99917 FORMAT('Error in F01CKF:berekening van V1TP3')
99916 FORMAT('Error in F01CKF:berekening van V2TP3')
99915 FORMAT('Error in F01CKF:berekening van AA3')
99914 FORMAT('Error in F01CKF:berekening van V1TP4')
99913 FORMAT('Error in F01CKF:berekening van V2TP4')
99912 FORMAT('Error in F01CKF:berekening van V3TP4')
99911 FORMAT('Error in F01CKF:berekening van AA4')
99910 FORMAT('Error in F01CKF:berekening van V1TP5')
99909 FORMAT('Error in F01CKF:berekening van V2TP5')
99908 FORMAT('Error in F01CKF:berekening van V3TP5')
99907 FORMAT('Error in F01CKF:berekening van V4TP5')
99906 FORMAT('Error in F01CKF:berekening van AA5')
99905 FORMAT('Error in F01CKF:berekening van V1TP6')
```

```
99904 FORMAT('Error in F01CKF:berekening van V2TP6')
99903 FORMAT('Error in F01CKF:berekening van V3TP6')
99902 FORMAT('Error in F01CKF:berekening van V4TP6')
99901 FORMAT('Error in F01CKF:berekening van V5TP6')
99900 FORMAT('Error in F01CKF:berekening van AA6')
C
      END
C
C ****************************************************************
C
      SUBROUTINE CALVIP
C     -----------------
C
      INTEGER M,N,NMINM,I,J,IFAIL
      DOUBLE PRECISION W4(1),W5(1),W8(1),W9(1),W10(1),W11(1),W13(1),
     %                 W14(1),W16(1),W17(1),W18(1),W19(1),W21(1),A2,
     %                 W22(1),W23(1),W31(1),W24(1),W26(1),W27(1),A3,
     %                 W28(1),W29(1),W30(1),W38(1),W40(1),W55(1),A4,
     %                 W32(1),W33(1),W34(1),W35(1),W37(1),W45(1),A5,
     %                 W46(1),W47(1),W41(1),W42(1),W43(1),W44(1),A6,
     %                 W48(1),W49(1),W51(1),W52(1),W54(1),A1,
     %                 PP(6,6),VP(6,6),P1PT(1,6),P3PT(1,6),GAMMA(4,6),
     %                 BP,VIP,P1P(6),V1P(6),V1PT(1,6),P1(6),
     %                 P1T(1,6),P2P(6),P2(6),V1T(1,6),V1B2(6),V1(6),
     %                 V1PA2(6),A2S(6),P2PT(1,6),P2T(1,6),V2P(6),
     %                 A22(6),P3P(6),P3(6),V1B31(6),V2PT(1,6),V2T(1,6),
     %                 V2B32(6),V2(6),V1PA3(6),V2PA3(6),A3S(6),
     %                 P3T(1,6),V3P(6),A33(6),P4P(6),P4(6),V1B41(6),
     %                 V2B42(6),V3PT(1,6),V3T(1,6),V3B43(6),V3(6),
     %                 V1PA4(6),V2PA4(6),V3PA4(6),A4S(6),P4PT(1,6),
     %                 P4T(1,6),V4P(6),P5P(6),P5(6),V1B51(6),V2B52(6),
     %                 V3B53(6),V4PT(1,6),V4T(1,6),V4B54(6),V4(6),
     %                 V1PA5(6),V2PA5(6),V3PA5(6),V4PA5(6),A5S(6),B2,
     %                 P5PT(1,6),P5T(1,6),V5P(6),A55(6),A44(6),P6P(6),
     %                 P6(6),V1B61(6),V2B62(6),V3B63(6),V4B64(6),
     %                 V5PT(1,6),V5T(1,6),V5B65(6),V5(6),V1PA6(6),
     %                 V2PA6(6),V3PA6(6),V4PA6(6),V5PA6(6),A6S(6),
     %                 P6PT(1,6),P6T(1,6),V6P(6),A66(6),A2P,GAMMAT(6,4),
     %                 BPT(6,2),A1PT,A1PN,A1P,V1PTP2,V1TP2P,V1TP2,
     %                 P2PTP2,A2PS,A2PT,P2TP2,A2PNS,A2PN,V1PTP3,B31,
     %                 V1TP3P,V2TP3P,B32,V1TP3,V2PTP3,V2TP3,P3PTP3,A3PS,
     %                 A3PT,P3TP3,A3PNS,A3PN,A3P,V1PTP4,V1TP4P,B41,
     %                 V2PTP4,V2TP4P,B42,V3PTP4,V3TP4P,B43,V1TP4,V2TP4,
     %                 V3TP4,P4PTP4,A4PS,A4PT,P4TP4,A4PNS,A4PN,A4P,
     %                 V1PTP5,V1TP5P,B51,V2PTP5,V2TP5P,B52,V3PTP5,
     %                 V3TP5P,B53,V4PTP5,V4TP5P,B54,V1TP5,V2TP5,V3TP5,
     %                 V4TP5,P5PTP5,A5PS,A5PT,P5TP5,A5PNS,A5PN,A5P,
     %                 V1PTP6,V1TP6P,B61,V2PTP6,V2TP6P,B62,V3PTP6,
     %                 V3TP6P,B63,V4PTP6,V4TP6P,B64,V5PTP6,V5TP6P,B65,
     %                 V1TP6,V2TP6,V3TP6,V4TP6,V5TP6,P6PTP6,A6PS,A6PT,
     %                 P6TP6,A6PNS,A6PN,A6P
C
      COMMON/BLOCK1/M,N,NMINM
      COMMON/BLOCK14/VIP(6,2)
      COMMON/BLOCK17/BP(2,6)
C
      DO 1200 I=1,N
         DO 1210 J=1,NMINM
            BPT(I,J)=BP(J,I)
            PP(I,M+J)=BPT(I,J)
 1210    CONTINUE
 1200 CONTINUE
C
      CALL CGAMMA
```

```
C
      DO 1220 I=1,N
         DO 1230 J=1,M
            GAMMAT(I,J)=GAMMA(J,I)
            PP(I,J)=GAMMAT(I,J)
 1230    CONTINUE
 1220 CONTINUE
C
      CALL VORT
C
C uitrekenen van VIP m.b.v een recursief schema beschreven in
C paragraaf 6.2.3, waarbij de stappen i=1,..,n geprogrammeerd worden
C
C ---------------------------stap 1: i=1---------------------------
C
      DO 1240 I=1,N
         P1P(I)=PP(I,1)
         P1PT(1,I)=PP(I,1)
 1240 CONTINUE
C
      IFAIL=0
      CALL F01CKF(A1PT,P1PT,P1,1,1,N,W4,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99895)
C
      IFAIL=0
      CALL F01CKF(A1PN,P1T,P1,1,1,N,W5,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99894)
C
      A1P=-A1PT/A1PN
      DO 1250 I=1,N
         V1P(I)=P1(I)*A1P+P1P(I)*A1
 1250 CONTINUE
C
      DO 1260 I=1,N
         VP(I,1)=V1P(I)
         V1PT(1,I)=VP(I,1)
 1260 CONTINUE
C ---------------------------stap 2: i=2---------------------------
      DO 1270 I=1,N
         P2P(I)=PP(I,2)
 1270 CONTINUE
C
      IFAIL=0
      CALL F01CKF(V1PTP2,V1PT,P2,1,1,N,W8,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99893)
C
      IFAIL=0
      CALL F01CKF(V1TP2P,V1T,P2P,1,1,N,W9,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99893)
C
      B2=V1PTP2+V1TP2P
      DO 1280 I=1,N
         V1B2(I)=V1(I)*B2
 1280 CONTINUE
C
      DO 1290 I=1,N
         V1PA2(I)=V1P(I)*V1TP2
 1290 CONTINUE
C
      DO 1300 I=1,N
         A2S(I)=P2P(I)-V1B2(I)-V1PA2(I)
 1300 CONTINUE
C
      DO 1305 I=1,N
```

```
         P2PT(1,I)=PP(I,2)
 1305 CONTINUE
C
      IFAIL=0
      CALL F01CKF(P2PTP2,P2PT,P2,1,1,N,W10,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99892)
C
      A2PS=V1TP2*B2
      A2PT=P2PTP2-A2PS
C
      IFAIL=0
      CALL F01CKF(P2TP2,P2T,P2,1,1,N,W11,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99891)
C
      A2PNS=P2TP2-((V1TP2)**2)
      A2PN=A2PNS*SQRT(A2PNS)
      A2P=-A2PT/A2PN
      DO 1310 I=1,N
         V2P(I)=A22(I)*A2P+A2S(I)*(1/A2)
 1310 CONTINUE
C
      DO 1320 I=1,N
         VP(I,2)=V2P(I)
 1320 CONTINUE
C ---------------------------stap 3: i=3--------------------------
      DO 1330 I=1,N
         P3P(I)=PP(I,3)
 1330 CONTINUE
C
      IFAIL=0
      CALL F01CKF(V1PTP3,V1PT,P3,1,1,N,W13,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99890)
C
      IFAIL=0
      CALL F01CKF(V1TP3P,V1T,P3P,1,1,N,W14,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99890)
C
      B31=V1PTP3+V1TP3P
      DO 1340 I=1,N
         V1B31(I)=V1(I)*B31
 1340 CONTINUE
C
      DO 1345 I=1,N
         V2PT(1,I)=VP(I,2)
 1345 CONTINUE
C
      IFAIL=0
      CALL F01CKF(V2PTP3,V2PT,P3,1,1,N,W16,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99889)
C
      IFAIL=0
      CALL F01CKF(V2TP3P,V2T,P3P,1,1,N,W17,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99889)
C
      B32=V2PTP3+V2TP3P
      DO 1350 I=1,N
         V2B32(I)=V2(I)*B32
 1350 CONTINUE
C
      DO 1360 I=1,N
         V1PA3(I)=V1P(I)*V1TP3
 1360 CONTINUE
C
      DO 1370 I=1,N
```

```
         V2PA3(I)=V2P(I)*V2TP3
 1370 CONTINUE
C
      DO 1380 I=1,N
         A3S(I)=P3P(I)-V1B31(I)-V1PA3(I)-V2B32(I)-V2PA3(I)
 1380 CONTINUE
C
      DO 1385 I=1,N
         P3PT(1,I)=PP(I,3)
 1385 CONTINUE
C
      IFAIL=0
      CALL F01CKF(P3PTP3,P3PT,P3,1,1,N,W18,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99888)
C
      A3PS=V1TP3*B31+V2TP3*B32
      A3PT=P3PTP3-A3PS
C
      IFAIL=0
      CALL F01CKF(P3TP3,P3T,P3,1,1,N,W19,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99887)
C
      A3PNS=P3TP3-((V1TP3)**2)-((V2TP3)**2)
      A3PN=A3PNS*SQRT(A3PNS)
      A3P=-A3PT/A3PN
      DO 1390 I=1,N
         V3P(I)=A33(I)*A3P+A3S(I)*(1/A3)
 1390 CONTINUE
C
      DO 1400 I=1,N
         VP(I,3)=V3P(I)
 1400 CONTINUE
C ---------------------------stap 4: i=4--------------------------
      DO 1410 I=1,N
         P4P(I)=PP(I,4)
 1410 CONTINUE
C
      IFAIL=0
      CALL F01CKF(V1PTP4,V1PT,P4,1,1,N,W21,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99886)
C
      IFAIL=0
      CALL F01CKF(V1TP4P,V1T,P4P,1,1,N,W22,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99886)
C
      B41=V1PTP4+V1TP4P
      DO 1420 I=1,N
         V1B41(I)=V1(I)*B41
 1420 CONTINUE
C
      IFAIL=0
      CALL F01CKF(V2PTP4,V2PT,P4,1,1,N,W23,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99885)
C
      IFAIL=0
      CALL F01CKF(V2TP4P,V2T,P4P,1,1,N,W24,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99885)
C
      B42=V2PTP4+V2TP4P
      DO 1430 I=1,N
         V2B42(I)=V2(I)*B42
 1430 CONTINUE
C
      DO 1435 I=1,N
```

```
         V3PT(1,I)=VP(I,3)
 1435 CONTINUE
C
      IFAIL=0
      CALL F01CKF(V3PTP4,V3PT,P4,1,1,N,W26,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99884)
C
      IFAIL=0
      CALL F01CKF(V3TP4P,V3T,P4P,1,1,N,W27,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99883)
C
      B43=V3PTP4+V3TP4P
      DO 1440 I=1,N
         V3B43(I)=V3(I)*B43
 1440 CONTINUE
C
      DO 1450 I=1,N
         V1PA4(I)=V1P(I)*V1TP4
 1450 CONTINUE
C
      DO 1460 I=1,N
         V2PA4(I)=V2P(I)*V2TP4
 1460 CONTINUE
C
      DO 1470 I=1,N
         V3PA4(I)=V3P(I)*V3TP4
 1470 CONTINUE
C
      DO 1480 I=1,N
         A4S(I)=P4P(I)-V1B41(I)-V1PA4(I)-V2B42(I)-V2PA4(I)-V3B43(I)
     %          -V3PA4(I)
 1480 CONTINUE
C
      DO 1485 I=1,N
         P4PT(1,I)=PP(I,4)
 1485 CONTINUE
C
      IFAIL=0
      CALL F01CKF(P4PTP4,P4PT,P4,1,1,N,W28,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99882)
C
      A4PS=V1TP4*B41+V2TP4*B42+V3TP4*B43
      A4PT=P4PTP4-A4PS
C
      IFAIL=0
      CALL F01CKF(P4TP4,P4T,P4,1,1,N,W29,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99881)
C
      A4PNS=P4TP4-((V1TP4)**2)-((V2TP4)**2)-((V3TP4)**2)
      A4PN=A4PNS*SQRT(A4PNS)
      A4P=-A4PT/A4PN
      DO 1490 I=1,N
         V4P(I)=A44(I)*A4P+A4S(I)*(1/A4)
 1490 CONTINUE
C
      DO 1500 I=1,N
         VP(I,4)=V4P(I)
 1500 CONTINUE
C ----------------------------stap 5: i=5--------------------------
      DO 1510 I=1,N
         P5P(I)=PP(I,5)
 1510 CONTINUE
C
      IFAIL=0
```

```
      CALL F01CKF(V1PTP5,V1PT,P5,1,1,N,W30,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99880)
C
      IFAIL=0
      CALL F01CKF(V1TP5P,V1T,P5P,1,1,N,W31,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99880)
C
      B51=V1PTP5+V1TP5P
      DO 1520 I=1,N
         V1B51(I)=V1(I)*B51
 1520 CONTINUE
C
      IFAIL=0
      CALL F01CKF(V2PTP5,V2PT,P5,1,1,N,W32,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99879)
C
      IFAIL=0
      CALL F01CKF(V2TP5P,V2T,P5P,1,1,N,W33,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99879)
C
      B52=V2PTP5+V2TP5P
      DO 1530 I=1,N
         V2B52(I)=V2(I)*B52
 1530 CONTINUE
C
      IFAIL=0
      CALL F01CKF(V3PTP5,V3PT,P5,1,1,N,W34,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99878)
C
      IFAIL=0
      CALL F01CKF(V3TP5P,V3T,P5P,1,1,N,W35,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99878)
C
      B53=V3PTP5+V3TP5P
      DO 1540 I=1,N
         V3B53(I)=V3(I)*B53
 1540 CONTINUE
C
      DO 1545 I=1,N
         V4PT(1,I)=VP(I,4)
 1545 CONTINUE
C
      IFAIL=0
      CALL F01CKF(V4PTP5,V4PT,P5,1,1,N,W37,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99877)
C
      IFAIL=0
      CALL F01CKF(V4TP5P,V4T,P5P,1,1,N,W38,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99877)
C
      B54=V4PTP5+V4TP5P
      DO 1550 I=1,N
         V4B54(I)=V4(I)*B54
 1550 CONTINUE
C
      DO 1560 I=1,N
         V1PA5(I)=V1P(I)*V1TP5
 1560 CONTINUE
C
      DO 1570 I=1,N
         V2PA5(I)=V2P(I)*V2TP5
 1570 CONTINUE
C
      DO 1580 I=1,N
```

```
         V3PA5(I)=V3P(I)*V3TP5
 1580 CONTINUE
C
      DO 1590 I=1,N
         V4PA5(I)=V4P(I)*V4TP5
 1590 CONTINUE
C
      DO 1600 I=1,N
         A5S(I)=P5P(I)-V1B51(I)-V1PA5(I)-V2B52(I)-V2PA5(I)-V3B53(I)
     %           -V3PA5(I)-V4B54(I)-V4PA5(I)
 1600 CONTINUE
C
      DO 1610 I=1,N
         P5PT(1,I)=PP(I,5)
 1610 CONTINUE
C
      IFAIL=0
      CALL F01CKF(P5PTP5,P5PT,P5,1,1,N,W40,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99876)
C
      A5PS=V1TP5*B51+V2TP5*B52+V3TP5*B53+V4TP5*B54
      A5PT=P5PTP5-A5PS
C
      IFAIL=0
      CALL F01CKF(P5TP5,P5T,P5,1,1,N,W41,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99875)
C
      A5PNS=P5TP5-((V1TP5)**2)-((V2TP5)**2)-((V3TP5)**2)-((V4TP5)**2)
      A5PN=A5PNS*SQRT(A5PNS)
      A5P=-A5PT/A5PN
      DO 1620 I=1,N
         V5P(I)=A55(I)*A5P+A5S(I)*(1/A5)
 1620 CONTINUE
C
      DO 1630 I=1,N
         VP(I,5)=V5P(I)
 1630 CONTINUE
C ---------------------------stap 6: i=6---------------------------
      DO 1640 I=1,N
         P6P(I)=PP(I,6)
 1640 CONTINUE
C
      IFAIL=0
      CALL F01CKF(V1PTP6,V1PT,P6,1,1,N,W42,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99874)
C
      IFAIL=0
      CALL F01CKF(V1TP6P,V1T,P6P,1,1,N,W43,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99874)
C
      B61=V1PTP6+V1TP6P
      DO 1650 I=1,N
         V1B61(I)=V1(I)*B61
 1650 CONTINUE
C
      IFAIL=0
      CALL F01CKF(V2PTP6,V2PT,P6,1,1,N,W44,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99873)
C
      IFAIL=0
      CALL F01CKF(V2TP6P,V2T,P6P,1,1,N,W45,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99873)
C
      B62=V2PTP6+V2TP6P
```

```
      DO 1660 I=1,N
         V2B62(I)=V2(I)*B62
 1660 CONTINUE
C
      IFAIL=0
      CALL F01CKF(V3PTP6,V3PT,P6,1,1,N,W46,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99872)
C
      IFAIL=0
      CALL F01CKF(V3TP6P,V3T,P6P,1,1,N,W47,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99872)
C
      B63=V3PTP6+V3TP6P
      DO 1670 I=1,N
         V3B63(I)=V3(I)*B63
 1670 CONTINUE
C
      IFAIL=0
      CALL F01CKF(V4PTP6,V4PT,P6,1,1,N,W48,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99871)
C
      IFAIL=0
      CALL F01CKF(V4TP6P,V4T,P6P,1,1,N,W49,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99871)
C
      B64=V4PTP6+V4TP6P
      DO 1680 I=1,N
         V4B64(I)=V4(I)*B64
 1680 CONTINUE
C
      DO 1690 I=1,N
         V5PT(1,I)=VP(I,5)
 1690 CONTINUE
C
      IFAIL=0
      CALL F01CKF(V5PTP6,V5PT,P6,1,1,N,W51,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99870)
C
      IFAIL=0
      CALL F01CKF(V5TP6P,V5T,P6P,1,1,N,W52,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99870)
C
      B65=V5TP6+V5TP6P
      DO 1700 I=1,N
         V5B65(I)=V5(I)*B65
 1700 CONTINUE
C
      DO 1710 I=1,N
         V1PA6(I)=V1P(I)*V1TP6
 1710 CONTINUE
C
      DO 1720 I=1,N
         V2PA6(I)=V2P(I)*V2TP6
 1720 CONTINUE
C
      DO 1730 I=1,N
         V3PA6(I)=V3P(I)*V3TP6
 1730 CONTINUE
C
      DO 1740 I=1,N
         V4PA6(I)=V4P(I)*V4TP6
 1740 CONTINUE
C
      DO 1750 I=1,N
```

```
         V5PA6(I)=V5P(I)*V5TP6
 1750 CONTINUE
C
      DO 1760 I=1,N
         A6S(I)=P6P(I)-V1B61(I)-V1PA6(I)-V2B62(I)-V2PA6(I)-V3B63(I)
     %           -V3PA6(I)-V4B64(I)-V4PA6(I)-V5B65(I)-V5PA6(I)
 1760 CONTINUE
C
      DO 1770 I=1,N
         P6PT(1,I)=PP(I,6)
 1770 CONTINUE
C
      IFAIL=0
      CALL F01CKF(P6PTP6,P6PT,P6,1,1,N,W54,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99869)
C
      A6PS=V1TP6*B61+V2TP6*B62+V3TP6*B63+V4TP6*B64+V5TP6*B65
      A6PT=P6PTP6-A6PS
C
      IFAIL=0
      CALL F01CKF(P6TP6,P6T,P6,1,1,N,W55,1,1,IFAIL)
      IF (IFAIL.GT.0) WRITE(1,99868)
C
      A6PNS=P6TP6-((V1TP6)**2)-((V2TP6)**2)-((V3TP6)**2)-((V4TP6)**2)
     %      -((V5TP6)**2)
      A6PN=A6PNS*SQRT(A6PNS)
      A6P=-A6PT/A6PN
      DO 1780 I=1,N
         V6P(I)=A66(I)*A6P+A6S(I)*(1/A6)
 1780 CONTINUE
C
      DO 1790 I=1,N
         VP(I,6)=V6P(I)
 1790 CONTINUE
C
      DO 1800 I=1,N
         DO 1810 J=1,NMINM
            VIP(I,J)=VP(I,M+J)
 1810    CONTINUE
 1800 CONTINUE
C
      RETURN
C
99895 FORMAT('Error in F01CKF:berekening van A1PT')
99894 FORMAT('Error in F01CKF:berekening van A1PN')
99893 FORMAT('Error in F01CKF:berekening van V1PTP2 of V1TP2P')
99892 FORMAT('Error in F01CKF:berekening van P2PTP2')
99891 FORMAT('Error in F01CKF:berekening van P2TP2')
99890 FORMAT('Error in F01CKF:berekening van V1PTP3 of V1TP3P')
99889 FORMAT('Error in F01CKF:berekening van V2PTP3 of V2TP3P')
99888 FORMAT('Error in F01CKF:berekening van P3PTP3')
99887 FORMAT('Error in F01CKF:berekening van P3TP3')
99886 FORMAT('Error in F01CKF:berekening van V1PTP4 of V1TP4P')
99885 FORMAT('Error in F01CKF:berekening van V2PTP4 of V2TP4P')
99884 FORMAT('Error in F01CKF:berekening van V3PTP4')
99883 FORMAT('Error in F01CKF:berekening van V3TP4P')
99882 FORMAT('Error in F01CKF:berekening van P4PTP4')
99881 FORMAT('Error in F01CKF:berekening van P4TP4')
99880 FORMAT('Error in F01CKF:berekening van V1PTP5 of V1TP5P')
99879 FORMAT('Error in F01CKF:berekening van V2PTP5 of V2TP5P')
99878 FORMAT('Error in F01CKF:berekening van V3PTP5 of V3TP5P')
99877 FORMAT('Error in F01CKF:berekening van V4PTP5 of V4TP5P')
99876 FORMAT('Error in F01CKF:berekening van P5PTP5')
99875 FORMAT('Error in F01CKF:berekening van P5TP5')
```

```
99874 FORMAT('Error in F01CKF:berekening van V1PTP6 of V1TP6P')
99873 FORMAT('Error in F01CKF:berekening van V2PTP6 of V2TP6P')
99872 FORMAT('Error in F01CKF:berekening van V3PTP6 of V3TP6P')
99871 FORMAT('Error in F01CKF:berekening van V4PTP6 of V4TP6P')
99870 FORMAT('Error in F01CKF:berekening van V5PTP6 of V5TP6P')
99869 FORMAT('Error in F01CKF:berekening van P6PTP6')
99868 FORMAT('Error in F01CKF:berekening van P6TP6')
C
      END
C
C ****************************************************************
C
```

BIJLAGE

Deze bijlage bevat de volgende routines uit de NAG-library:

- **D02CAF:** integreert een eerste orde differentiaalvgl., met gebruik van een variabele orde en variabele stapgrootte integratieroutine
- **F01AAF:** berekent de inverse van een reële matrix
- **F01CKF:** berekent het product van twee matrices
- **F04ATF:** berekent de oplossing van een stelsel Ax=b

D02CAF – NAG Fortran Library Routine Document

NOTE: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details. The routine name may be precision-dependent.

Warning: the specification of the parameters XEND, TOL and W have changed at Mark 13: W is now an array of one dimension.

1. Purpose

D02CAF integrates a system of first-order ordinary differential equations over a range with suitable initial conditions, using a variable-order variable-step Adams method.

2. Specification

```
      SUBROUTINE D02CAF (X, XEND, N, Y, TOL, FCN, W,
     1           IFAIL)
C     INTEGER    N, IFAIL
C     real       X, XEND, Y(N), TOL, W(23+21*N)
C     EXTERNAL   FCN
```

3. Description

The routine integrates a system of ordinary differential equations

$$Y_i' = F_i(T,Y_1,Y_2,...,Y_N) \qquad i = 1,2,...,N$$

from T = X to T = XEND using a variable-order variable-step Adams method. The system is defined by a subroutine FCN supplied by the user, which evaluates F_i in terms of T and $Y_1,Y_2,...,Y_N$ (see Section 5), and the values of $Y_1,Y_2,...,Y_N$ must be given at T = X. The accuracy of the integration is controlled by the parameter TOL.

For a description of Adams methods and their practical implementation see [1].

4. References

[1] HALL, G. and WATT, J.M. (eds).
Modern Numerical Methods for Ordinary Differential Equations.
Clarendon Press, Oxford, 1976.

5. Parameters

X – ***real.***

Before entry, X must be set to the initial value of the independent variable T.

On exit, it contains XEND, unless an error has occurred, when it contains the value of T at the error.

XEND – ***real.***

On entry, XEND must specify the final value of the independent variable. If XEND < X on entry, integration will proceed in the negative direction.

XEND ≠ X.

Unchanged on exit.

N – INTEGER.

On entry, N must specify the number of differential equations.

Unchanged on exit.

Y – ***real*** array of DIMENSION at least (N).

Before entry, Y(1),Y(2),...,Y(N) must contain the initial values of the solution $Y_1,Y_2,...,Y_N$.

On exit, Y(1),Y(2),...,Y(N) contain the computed values of the solution at the final value of T.

TOL – ***real.***

Before entry, TOL must specify a **positive** tolerance for controlling the error in the integration.

D02CAF has been designed so that for most problems a reduction in TOL leads to an approximately proportional reduction in the error in the solution at XEND. However, the actual relation between TOL and the accuracy achieved cannot be guaranteed. The user is strongly recommended to call D02CAF with more than one value for TOL and to compare the results obtained to estimate their accuracy. In the abscence of any prior knowledge, the user might compare the results obtained by calling D02CAF with TOL = 10.0^{-P} and TOL = 10.0^{-P-1} if P correct decimal digits in the solution are required.

Unchanged on exit.

FCN – SUBROUTINE, supplied by the user.

FCN must evaluate the functions F_i (i.e. the derivatives Y'_i) for given values of its arguments $T, Y_1, ..., Y_N$.

Its specification is:

```
SUBROUTINE   FCN(T, Y, F)
real         T, Y(n), F(n)
```

where n is the actual value of N in the call of D02CAF.

T – ***real.***

On entry, T specifies the value of the argument T.

Its value must not be changed.

Y – ***real*** array of DIMENSION (n).

On entry, Y(I) contains the value of the argument Y_I, for I = 1,...,n.

These values must not be changed.

F – ***real*** array of DIMENSION (n).

On exit, F(I) must contain the value of F_I, for I = 1,...,n.

FCN must be declared as EXTERNAL in the (sub)program from which D02CAF is called.

W – ***real*** array of DIMENSION (23+21*N).

Used as working space.

IFAIL – INTEGER.

Before entry, IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

Unless the routine detects an error (see Section 6), IFAIL contains 0 on exit.

6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or –1, explanatory error messages are output on the current error message unit (as defined by NAG Fortran Library routine X04AAF).

IFAIL = 1

On entry, TOL ≤ 0.0,
or X = XEND,
or N ≤ 0.

The last error will cause a program breakdown with some compilers.

IFAIL = 2

With the given value of TOL, no further progress can be made across the integration range from the current point T = X, or the dependence of the error on TOL would be lost if further progress across the integration range were attempted (see Section 8 for a discussion of this error exit). The components Y(1),Y(2),...,Y(N) contain the computed values of the solution at the current point T = X.

IFAIL = 3

TOL is too small for the routine to start the integration (see Section 8). X and Y(1),Y(2),...,Y(N) retain their initial values.

IFAIL = 4

A serious error has occurred in an internal call. Check all subroutine calls and array dimensions. Seek expert help.

7. Accuracy

The accuracy depends on TOL, on the mathematical properties of the differential system and on the length of the range of integration. It can be controlled by varying TOL but the approximate proportionality of the error to TOL holds only for a restricted range of values of TOL. For TOL too large, the underlying theory may break down and the result of varying TOL may be unpredictable. For TOL too small, rounding errors may affect the solution significantly and an error exit with IFAIL = 2 or IFAIL = 3 is possible.

The user who requires a more reliable estimate of the accuracy achieved than can be obtained by varying TOL, is recommended to call the NAG Fortran Library routine D02BDF where both the solution and a global error estimate are computed.

8. Further Comments

Timing depends on the complexity and mathematical properties of the system of differential equations defined by FCN, on the length of the range, and on the tolerance. There is also a small overhead of the form A + B×N, where A and B are machine-dependent computing times.

If the routine fails with IFAIL = 3, then it could be called again with a larger value of TOL if this has not already been tried. If the accuracy requested is really needed and cannot be obtained with this routine, then the system may be very stiff (see below) or so badly scaled that it cannot be solved to the required accuracy.

If the routine fails with IFAIL = 2, it is probable that it has been called with a value of TOL which is so small that a solution cannot be obtained on the range X to XEND. This can happen for well-behaved systems and very small values of TOL. The user should, however, consider whether there is a more fundamental difficulty. For example:

(a) in the region of a singularity (infinite value) of the solution, the routine will usually stop with IFAIL = 2, unless overflow occurs first. If overflow occurs using D02CAF, NAG Fortran Library routine D02QFF can be used instead to trap the increasing solution before overflow occurs. In any case, numerical integration cannot be continued through a singularity, and analytical treatment should be considered;

(b) for 'stiff' equations, where the solution contains rapidly decaying components, the routine will use very small steps in T (internally to D02CAF) to preserve stability. This will exhibit itself by making the computing time excessively long, or occasionally by an exit with IFAIL = 2. Adams methods are not efficient in such cases and the user should try the NAG Fortran Library Backward Differentiation Formula method routine D02EAF.

Users with problems for which D02CAF is not sufficiently general should consider the NAG Fortran Library routines D02CJF, and D02QFF. Routine D02CJF can be used when output is required as points inside the range X to XEND (for example, for graph-plotting purposes) or more general error control is required. Also, use of routine D02CJF should be computationally more efficient than repeated calls to D02CAF to achieve the same result and can be used to calculate where a function of the components $Y_1, Y_2, \ldots, Y_N$ and their derivatives takes a specified value.

Routine D02QFF is a more general Adams routine with many facilities including more general error control options and several criteria for interrupting the calculations, by use of a root-finding option.

9. Example

To integrate the following equations (for a projectile)

$$y' = \tan(\phi)$$
$$v' = -0.032 \times \tan(\phi)/v - 0.02 \times v/\cos(\phi)$$
$$\phi' = -0.032/v^2$$

over an interval X = 0.0 to XEND = 8.0 starting with values y = 0.0, v = 0.5 and $\phi = \pi/5$. We write y = Y(1), v = Y(2) and ϕ = Y(3) and we set TOL = 1.0E–4 and TOL = 1.0E–5 in turn so that we may compare the solutions obtained. The value of π is obtained by using NAG Fortran Library routine X01AAF.

9.1. Program Text

NOTE: The listing of the example program presented below uses ***bold italicised*** terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
C     D02CAF EXAMPLE PROGRAM TEXT
C     MARK 13 REVISED.  NAG COPYRIGHT 1988.
C     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          N
      PARAMETER        (N=3)
C     .. Local Scalars ..
      real             PI, TOL, X, XEND
      INTEGER          I, IFAIL, J
C     .. Local Arrays ..
      real             W(21*N+23), Y(N)
C     .. External Functions ..
      real             X01AAF
      EXTERNAL         X01AAF
C     .. External Subroutines ..
      EXTERNAL         D02CAF, FCN
C     .. Executable Statements ..
      WRITE (NOUT,FMT=99996)
      PI = X01AAF(0.0e0)
      XEND = 8.0e0
      DO 20 J = 4, 5
         TOL = 10.0e0**(-J)
         X = 0.0e0
         Y(1) = 0.0e0
         Y(2) = 0.5e0
         Y(3) = PI/5.0e0
         IFAIL = 0
         WRITE (NOUT,FMT=99998) TOL
         WRITE (NOUT,FMT=99999)
         WRITE (NOUT,FMT=99997) X, (Y(I),I=1,N)
         CALL D02CAF(X,XEND,N,Y,TOL,FCN,W,IFAIL)
         WRITE (NOUT,FMT=99997) X, (Y(I),I=1,N)
   20 CONTINUE
      STOP
C
```

F01AAF – NAG FORTRAN Library Routine Document

NOTE: before using this routine, please read the appropriate implementation document to check the interpretation of ***bold italicised*** terms and other implementation–dependent details. The routine name may be precision–dependent.

1. Purpose

F01AAF calculates the approximate inverse of a real matrix by Crout's method.

2. Specification

```
      SUBROUTINE F01AAF (A, IA, N, UNIT, IUNIT, WKSPCE, IFAIL)
C     INTEGER IA, N, IUNIT, IFAIL
C     real A(IA,N), UNIT(IUNIT,N), WKSPCE(N)
```

3. Description

The routine calculates the inverse X of a real matrix A by solving the set of linear equations AX = I using Crout's method with partial pivoting and ***additional precision*** accumulation of innerproducts. The matrix A, provided it is non-singular, is factorised into the form PA = LU, where P is a permutation matrix, L is lower triangular and U is unit upper triangular. The columns of the inverse X are found by forward and backward substitution in the equations Ly = Pe and Ux = y, where e is a column of the identity matrix I.

4. References

[1] WILKINSON, J.H. and REINSCH, C. Handbook for Automatic Computation. Volume II, Linear Algebra, pp. 93–110. Springer–Verlag, 1971.

5. Parameters

A – ***real*** array of DIMENSION (IA,p) where $p \geq N$.

Before entry, A must contain the elements of the real matrix.

On successful exit, it contains the Crout factorisation with the unit diagonal of U understood.

IA – INTEGER.

On entry, IA must specify the first dimension of array A as declared in the calling (sub)program.
$IA \geq N$.

Unchanged on exit.

N – INTEGER.

On entry, N must specify the order of matrix A.

Unchanged on exit.

UNIT – ***real*** array of DIMENSION (IUNIT,p) where $p \geq N$.

On successful exit, UNIT contains the inverse of A.

IUNIT – INTEGER.

On entry, IUNIT must specify the first dimension of array UNIT as declared in the calling (sub)program.
$IUNIT \geq N$.

Unchanged on exit.

WKSPCE – ***real*** array of DIMENSION at least (N).

Used as working space.

IFAIL – INTEGER.

On entry, IFAIL must be set to 0 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

Unless the routine detects an error (see next section), IFAIL contains 0 on exit.

6. Error Indicators and Warnings

Errors detected by the routine:–

IFAIL = 1

The matrix A is singular or almost singular, possibly due to rounding errors.

7. Auxiliary Routines

Details are distributed to sites in machine-readable form.

8. Timing

The time taken is approximately proportional to N^3.

9. Storage

There are no internally declared arrays.

10. Accuracy

The accuracy of the computed inverse depends on the conditioning of the original matrix. For a detailed error analysis see [1], page 107.

11. Further Comments

The accurate inverse of a real matrix can be found using F04AEF.

12. Keywords

Approximate Inverse,
Crout Factorisation,
Real Matrix.

---

13. Example

To find the inverse of the 3 × 3 matrix:

$$\begin{pmatrix} 33 & 16 & 72 \\ -24 & -10 & -57 \\ -8 & -4 & -17 \end{pmatrix}$$

13.1. Program Text

WARNING: This **single precision** example program may require amendment for certain implementations. The results produced may not be the same. If in doubt, please seek further advice (see **Essential Introduction** to the Library Manual).

```
C     F01AAF EXAMPLE PROGRAM TEXT
C     NAG COPYRIGHT 1975
C     MARK 4.5 REVISED
C
      REAL A(5,5), UNIT(5,5), WKSPCE(7)
      INTEGER NIN, NOUT, I, N, J, IA, IUNIT, IFAIL
      DATA NIN /5/, NOUT /6/
      READ (NIN,99999) (WKSPCE(I),I=1,7)
      WRITE (NOUT,99997) (WKSPCE(I),I=1,6)
      N = 3
      READ (NIN,99998) ((A(I,J),J=1,N),I=1,N)
      IA = 5
      IUNIT = 5
      IFAIL = 1
      CALL F01AAF(A, IA, N, UNIT, IUNIT, WKSPCE, IFAIL)
      IF (IFAIL.EQ.0) GO TO 20
      WRITE (NOUT,99996) IFAIL
      STOP
   20 WRITE (NOUT,99995) ((UNIT(I,J),J=1,N),I=1,N)
      STOP
99999 FORMAT (6A4, 1A3)
99998 FORMAT (3F5.0)
99997 FORMAT (4(1X/), 1H , 5A4, 1A3, 7HRESULTS/1X)
99996 FORMAT (25H0ERROR IN F01AAF IFAIL = , I2)
99995 FORMAT (8H0INVERSE/(1H , 3F10.4))
      END
```

F01CKF

1. Purpose

F01CKF returns with the result of the multiplication of two matrices B and C in the matrix A, with the option to overwrite B or C.

IMPORTANT: before using this routine, read the appropriate machine implementation document to check the interpretation of italicised terms and other implementation-dependent details.

2. Specification (FORTRAN IV)

```
      SUBROUTINE F01CKF(A,B,C,N,P,M,Z,IZ,OPT,IFAIL)
C     INTEGER N,P,M,IZ,OPT,IFAIL
C     real A,B,C,Z
C     DIMENSION A(N,P),B(N,M),C(M,P),Z(IZ)
```

3. Description

The $n \times m$ matrix B is post-multiplied by the $m \times p$ matrix C. If OPT=1 the result is formed in the $n \times p$ matrix A. If OPT=2, m must equal p, and the result is written back to B. If OPT=3, n must equal m, and the result is written back to C.

4. References

None.

5. Parameters

A - *real* array of DIMENSION (N,P).
On exit, if OPT=1, A contains the result of the matrix multiplication.

B - *real* array of DIMENSION (N,M).
Before entry, all elements of B must be assigned a value.
On exit, if OPT=2, B contains the result of the multiplication.
Otherwise B is unchanged on exit.

C - *real* array of DIMENSION (M,P).
Before entry, all elements of C must be assigned a value.
On exit, if OPT=3, C contains the result of the multiplication.
Otherwise C is unchanged on exit.

N - INTEGER.
On entry, N specifies n, the first dimension of A and B as declared in the calling (sub)program. If OPT=3, n must equal m. Unchanged on exit.

P - INTEGER.
On entry, P specifies p, the second dimension of A and C as declared in the calling (sub)program. If OPT=2, p must equal m. Unchanged on exit.

F01CKF

5. Parameters (contd)

M - INTEGER.
On entry, M specifies m, the second dimension of B and first dimension of C as declared in the calling (sub)program. Unchanged on exit.

Z - *real* array of DIMENSION (IZ).
Used as working space.

IZ - INTEGER.
On entry, IZ specifies the dimension of Z. If OPT=1, IZ may be 1 otherwise IZ must be greater than or equal to m. Unchanged on exit.

OPT - INTEGER.
On entry, the value of OPT determines which array is to contain the final result.

a) OPT=1. A must be distinct from B and C and, on exit, contains the result. B and C need not be distinct in this case.

b) OPT=2. B must be distinct from C and on exit, contains the result. A is not used in this case and need not be distinct from B or C.

c) OPT=3. C must be distinct from B and on exit, contains the result. A is not used in this case and need not be distinct from B or C.

OPT is unchanged on exit.

IFAIL - INTEGER.
Before entry, IFAIL must be assigned a value. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0. Unless the routine detects an error (see Section 6), IFAIL contains 0 on exit.

6. Error Indicators

Errors detected by the routine:-

IFAIL = 1 On entry, M or P or N $\leq$ 0.

IFAIL = 2 OPT=2 and M $\neq$ P.

IFAIL = 3 OPT=3 and N $\neq$ M.

IFAIL = 4 OPT$\neq$1 and IZ < M.

7. Auxiliary Routines

This routine calls the NAG Library routine P01AAF.

8. Timing

The timing increases with m,p and n.

9. Storage

There are no internally declared arrays.

10. Accuracy

Inner-products are accumulated using *additional precision.*

11. Further Comments None.

12. Keywords

Matrix Multiplication.

13. Example

The example program multiplies the 2 × 3 matrix B and the 3 × 2 matrix C together and places the result in the 2 × 2 matrix A.

Program

This single precision example program may require amendment
i) for use in a DOUBLE PRECISION implementation
ii) for use in either precision in certain implementations.
The results produced may differ slightly.

```
C     F01CKF EXAMPLE PROGRAM TEXT
C     NAG COPYRIGHT 1975
C     MARK 4.5 REVISED
      REAL A(2,2), B(2,3), C(3,2), Z(1)
      INTEGER I, IFAIL, NOUT
      DATA NOUT /6/
      WRITE (NOUT,99999)
      DO 20 I=1,3
         B(1,I) = FLOAT(I) - 1.
         C(I,1) = B(1,I)
         B(2,I) = FLOAT(I)
         C(I,2) = B(2,I)
   20 CONTINUE
      IFAIL = 0
      CALL F01CKF(A, B, C, 2, 2, 3, Z, 1, 1, IFAIL)
      IF (IFAIL.GT.0) GO TO 40
      WRITE (NOUT,99998)
      WRITE (NOUT,99997) A(1,1), A(1,2), A(2,1), A(2,2)
      STOP
```

F04ATF – NAG FORTRAN Library Routine Document

NOTE: before using this routine, please read the appropriate implementation document to check the interpretation of ***bold italicised*** terms and other implementation–dependent details. The routine name may be precision–dependent.

1. Purpose

F04ATF calculates the accurate solution of a set of real linear equations with a single right hand side, Ax=b, by Crout's factorisation method.

2. Specification

```
      SUBROUTINE F04ATF (A, IA, B, N, C, AA, IAA, WKS1, WKS2,
     1   IFAIL)
C     INTEGER IA, N, IAA, IFAIL
C     real A(IA,N), B(N), C(N), AA(IAA,N), WKS1(N), WKS2(N)
```

3. Description

Given a set of linear equations, Ax = b, the routine first decomposes A using Crout's factorisation with partial pivoting PA = LU, where P is a permutation matrix, L is lower triangular and U is unit upper triangular. An approximation to x is found by forward and backward substitution in Ly = Pb and Ux = y. The residual vector r = b – Ax is then calculated and a correction, d, to x is found by the solution of LUd = r. x is replaced by (x+d) and the process repeated until full machine accuracy is obtained. ***Additional precision*** accumulation of innerproducts is used throughout the calculation.

4. References

[1] WILKINSON, J.H. and REINSCH, C. Handbook for Automatic Computation. Volume II, Linear Algebra. Springer–Verlag, 1971, pp. 93–110.

5. Parameters

A – ***real*** array of DIMENSION (IA,p) where p ≥ N.

Before entry, A must contain the elements of the real matrix.

Unchanged on exit.

IA – INTEGER.

On entry, IA must specify the first dimension of array A as declared in the calling (sub)program.
IA ≥ N.

Unchanged on exit.

B – ***real*** array of DIMENSION at least (N).

Before entry, B must contain the elements of the right hand side. (See Section 11).

Unchanged on exit.

N – INTEGER.

On entry, N must specify the order of matrix A.

Unchanged on exit.

C – ***real*** array of DIMENSION at least (N).

On successful exit, C will contain the solution vector.

AA – ***real*** array of DIMENSION (IAA,q) where q ≥ N.

Used as working space.

On successful exit, AA will contain the LU decomposition.

IAA – INTEGER.

On entry, IAA must specify the first dimension of array AA as declared in the calling (sub)program.
IAA ≥ N.

Unchanged on exit.

WKS1 – ***real*** array of DIMENSION at least (N).

WKS2 – ***real*** array of DIMENSION at least (N).

Used as working space.

IFAIL – INTEGER.

On entry, IFAIL must be set to 0 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

Unless the routine detects an error (see next section), IFAIL contains 0 on exit.

6. Error Indicators and Warnings

Errors detected by the routine:–

IFAIL = 1

The matrix A is singular, possibly due to rounding errors.

IFAIL = 2

The matrix A is too ill-conditioned to produce a correctly rounded solution.

7. Auxiliary Routines

Details are distributed to sites in machine-readable form.

8. Timing

The time taken is approximately proportional to N^3.

9. Storage

There are no internally declared arrays.

10. Accuracy

The computed solutions should be correct to full machine accuracy. For a detailed error analysis see [1], page 107.

11. Further Comments

The routine **must not** be called with the same name for parameters B and C.

12. Keywords

Accurate Solution of Linear Equations,
Crout Factorisation,
Real Matrix,
Single Right Hand Side.

---

13. Example

To solve the set of linear equations Ax = b where

$$A = \begin{pmatrix} 33 & 16 & 72 \\ -24 & -10 & -57 \\ -8 & -4 & -17 \end{pmatrix} \text{ and } b = \begin{pmatrix} -359 \\ 281 \\ 85 \end{pmatrix}$$

13.1. Program Text

WARNING: This **single precision** example program may require amendment for certain implementations. The results produced may not be the same. If in doubt, please seek further advice (see **Essential Introduction** to the Library Manual).

```
C     F04ATF EXAMPLE PROGRAM TEXT
C     NAG COPYRIGHT 1975
C     MARK 4.5 REVISED
C
      REAL A(5,5), B(5), C(5), AA(5,5), WKS1(18), WKS2(3)
      INTEGER NIN, NOUT, I, N, J, IA, IAA, IFAIL
      DATA NIN /5/, NOUT /6/
      READ (NIN,99999) (WKS1(I),I=1,7)
      WRITE (NOUT,99997) (WKS1(I),I=1,6)
      N = 3
      READ (NIN,99998) ((A(I,J),J=1,N),I=1,N), (B(I),I=1,N)
      IA = 5
      IAA = 5
      IFAIL = 1
      CALL F04ATF(A, IA, B, N, C, AA, IAA, WKS1, WKS2, IFAIL)
      IF (IFAIL.EQ.0) GO TO 20
      WRITE (NOUT,99996) IFAIL
      STOP
   20 WRITE (NOUT,99995) (C(I),I=1,N)
      STOP
99999 FORMAT (6A4, 1A3)
```