Wolfhard Zahlten

Vorlesungsreihe:

Numerische Methoden im Bauingenieurwesen

FEM III: Nichtlineare Probleme

Vorlesung 4

Pfadverfolgungsalgorithmen

Teil A: Theorie

MENUM

Überblick

Alle Algorithmen zur direkten Lösung nichtlinearer Gleichungssysteme basieren auf der wiederholten Lösung linearer Gleichungssysteme: die *Lastgeschichte* wird in eine Anzahl von *Lastschritten* eingeteilt, in die jeweils ein *Iterationszyklus* eingebettet ist. Würde beispielsweise die Gesamtlast in 20 Schritten aufgebracht, die jeweils im Schnitt 6 Iterationen benötigten, ergäben sich 20*(1+6)=140 Gleichungssysteme. Diese müssten *aufgebaut* und *gelöst* werden.

Der Berechnungsaufwand ist also im Nichtlinearen wesentlich größer als im Linearen. Somit stellen sich einige Fragen:

1. **Wie kann man eine möglichst geringe Rechenzeit bei gleichzeitig geringem Speicherbedarf erreichen?**
2. **Welche Gleichungslöser und Speichertechniken sind hierfür besonders geeignet?**

Das sind Fragen der numerischen Mathematik. Wir werden diese Aspekte nur soweit anreißen, wie es zur intelligenten Bedienung von nichtlinearen Programmpaketen notwendig ist. Beginnen werden wir mit einigen Gedanken zu *Speichertechniken* und *Lösungsalgorithmen* für lineare Gleichungssysteme.

Matrizen, Gleichungslöser etc ...

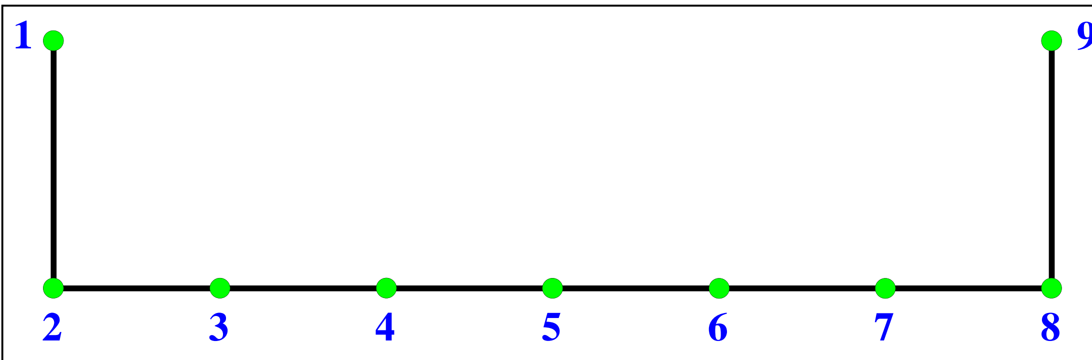
Belegungsschema der Steifigkeitsmatrix

Eine Systemsteifigkeitsmatrix weist nur dort Einträge ungleich Null auf, wo Knoten durch Elemente miteinander verbunden sind. Somit besitzt sie, unabhängig von den Zahlenwerten der Matrixeinträge, ein ***Belegungsschema***, welches ausschließlich von der ***Inzidenzverknüpfung***, also der ***Tragwerkstopologie***, abhängt.

Dieses ***Belegungsschema*** ist immer ***symmetrisch***, denn wenn der Knoten j mit dem Knoten k verbunden ist, ist automatisch auch der Knoten k mit dem Knoten j verbunden. Die ***Matrix*** selbst kann je nach Problemstellung ***symmetrisch*** oder ***unsymmetrisch belegt*** sein.

Das Belegungsschema ist für alle Steifigkeitsmatrizen (lineare Matrix, geometrische Matrix, tangentiale Matrix …) gleich.

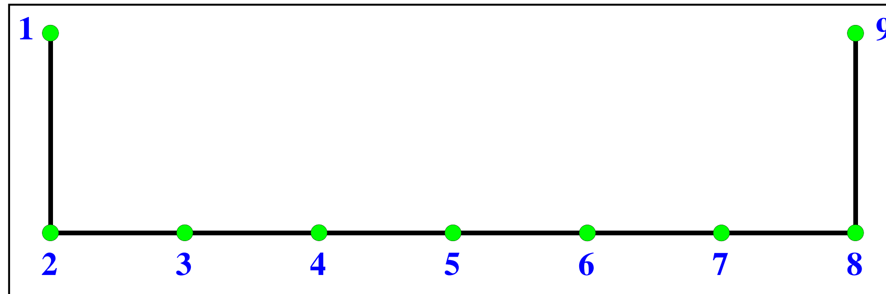
Beispiel: Linientragwerk

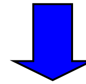


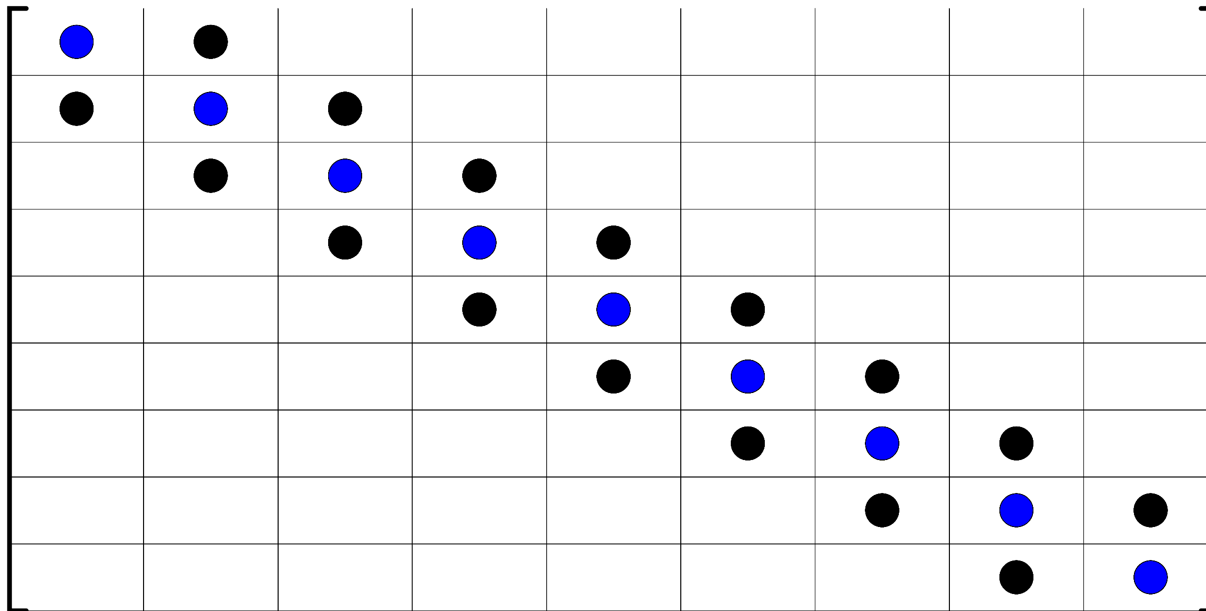
Wir nehmen an, dass jeder Knoten nur einen Freiheitsgrad hat, so dass die Systemmatrix eine 9x9-Matrix darstellt.

MENUM

Belegungsschema für das Beispiel







MENUM

Speicherschema I: allgemeine Vollmatrix

Die einfachste Speichertechnik speichert die gesamte Matrix unabhängig vom Belegungsschema ab. Der Speicherplatzbedarf S_d für die Matrixelemente ist damit maximal groß. Der Index d bei S zeigt an, dass es sich hierbei um reelle Zahlen handelt, die mittels des Datentyps „double“ erfasst werden, für den 8 Byte benötigt werden. Im Gegensatz dazu stehen ganze Zahlen, für die 4 Byte (32-Bit-Betriebssysteme) oder 8 Byte (64-Bit-Betriebssysteme) verwendet werden.

$$S_d = N \cdot N = 81$$

Speicherschema II: symmetrische Vollmatrix

Bei linearen Problemen ist die Steifigkeitsmatrix symmetrisch. Damit muss nur das obere oder untere Dreieck abgespeichert werden. Das führt zu einer Speicherplatzreduzierung auf fast die Hälfte.

Bei nichtlinearen Problemen kann die Matrix unsymmetrisch werden. Zur Speicherplatzreduktion werden deshalb oft Symmetrisierungen vorgenommen. Der dadurch entstehende Symmetrisierungsfehler kann dann iterativ eliminiert werden.

$$S_d = \frac{1}{2} N \cdot (N+1) = 45$$

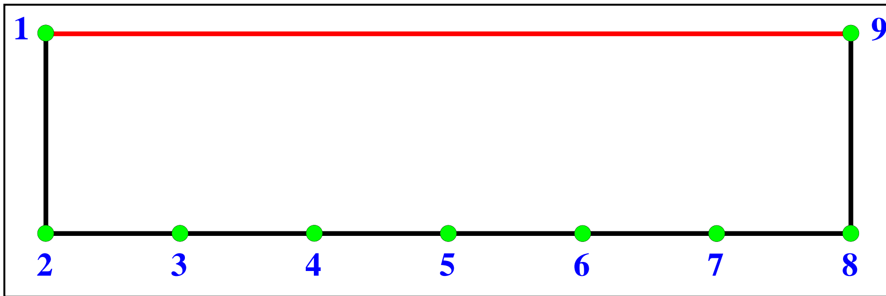
Speicherschema III: symmetrische Bandmatrix

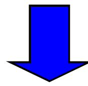
Das Belegungsschema zeigt eine deutliche *Bandstruktur*. Die Hauptdiagonale ist immer besetzt, aber ab einem gewissen Abstand, der *Bandbreite B*, sind alle weiter außen liegenden Elemente Null. Diese müssen nicht abgespeichert werden, so dass anstatt einer Quadratmatrix NxN eine Rechteckmatrix NxB abgelegt wird. Eventuelle Nullen innerhalb des Bandes (hier nicht vorhanden) werden mit abgespeichert. Am Ende der Matrix entsteht ein „Zwickel", für den Speicherplatz allokiert wird, der aber nicht verwendet wird.

B

$$S_d = N \cdot B = 9 \cdot 2 = 18$$

Ungeschickte Inzidenzverküpfung



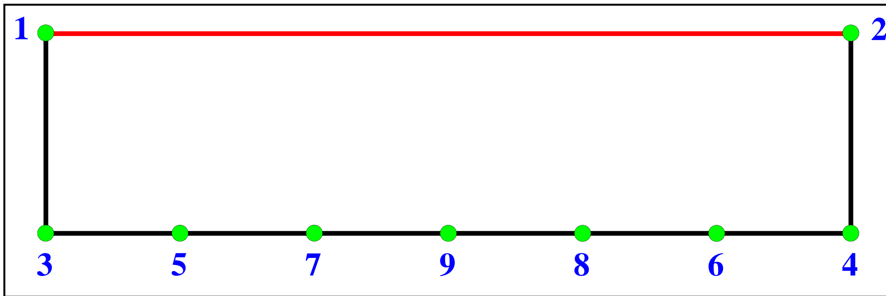


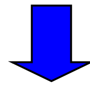
Durch Einfügen des roten Stabs verknüpft sich jetzt der Knoten 1 mit dem Knoten 9. Als Folge steigt die Bandbreite B auf den Maximalwert von N, also auf 9. Dadurch wird der ungenutzte „Zwickel“ so groß, dass der Speicherplatz dem der unsymmetrischen Vollmatrix entspricht.

$$S_d = N \cdot B = 9 \cdot 9 = 81$$

Diese Knotennummerierung ist also sehr ungünstig. Durch eine geschickte Umnummerierung kann die Bandbreite jedoch gesenkt werden.

Bandbreitenoptimierung





Die Tatsache, dass das Tragwerk jetzt zusammenhängend ist, kann auch durch eine Umnummerierung nicht behoben werden. Jedoch kann durch die dargestellte Nummerierung eine Bandbreitenreduktion auf 3 erreicht werden.

$$S_d = N \cdot B = 9 \cdot 3 = 27$$

Es gibt spezielle Algorithmen für eine sog. *Bandbreitenoptimierung*. Eine wirklich minimale Bandbreite finden diese nicht, aber es kann eine deutlich reduzierte Bandbreite in der Nähe des Optimums erreicht werden.

Speichertechnik IV: Skyline

Eine Verbesserung der Bandabspeicherung stellt die sog. ***„Skyline"-Technik*** dar. Es wird keine Bandbreite für die ganze Matrix festgelegt, sondern jede Zeile erhält ihre eigene Bandbreite. Somit werden horizontale Säulen abgespeichert. Man sieht, dass die unglückliche Verknüpfung des Knotens 1 mit 9 nur zu einer einzigen langen Säule in Zeile 1 wird, aber sonst keine negativen Auswirkungen besitzt. Die abzuspeichernden Zahlenwerte gehorchen jedoch keinem einfachen Schema mehr und werden hintereinander auf einem eindimensionalen Feld abgelegt.

$S_d = 24$

$S_i = 9$

1	10	12	14	16	18	20	22	24

Um aus diesen strukturlosen Daten wieder die Matrix herstellen zu können, wird ein ***Adressvektor*** benötigt, auf dem die Adressen der Diagonalelemente abgelegt sind. Damit kommen zu den eigentlichen Daten weitere ***Verwaltungsdaten*** hinzu, die ebenfalls Speicherplatz benötigen. Die Matrixelemente der 3. Zeile befinden sich damit auf den Plätzen adr(3) bis adr(4)-1, also auf 12 bis 13.

Speichertechnik V: Sparse-Matrix

Eine weitere Optimierung kann erreicht werden, wenn die Nulleinträge innerhalb jeder Säule ebenfalls nicht abgespeichert werden. Dann reduziert sich die Datenspeicherung auf die Nichtnullelemente allein.

$S_d = 18$

$S_i = 2 \cdot 18 = 36$

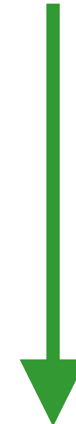
Adressen

1	1	1	2	2	3	...
1	2	9	2	3	3	...

Zu jedem Nichtnullelement gehören jetzt als Adressen dessen Zeilen- und Spaltenzahl. Wenn man 4 Byte für eine Adresse verwendet, verbrauchen die Adressen genauso viel Speicher wie die Werte selbst. Bei 8 Byte pro Integer benötigen die Adressen sogar den doppelten Platz wie die eigentlichen Daten. Somit ist diese Speichertechnik bei Vollmatrizen schlecht, bei sehr dünn besetzten Matrizen jedoch unschlagbar effizient.

Gleichungslösung bei einer Dreiecksmatrix

$$\begin{bmatrix} K_{1,1} & 0 & 0 & \cdots & 0 \\ K_{2,1} & K_{2,2} & 0 & \cdots & 0 \\ K_{3,1} & K_{3,2} & K_{3,3} & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ K_{N,1} & K_{N,2} & K_{N,3} & \cdots & K_{N,N} \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ \bullet \\ V_N \end{bmatrix} = \begin{bmatrix} P_1 \\ P_2 \\ P_2 \\ \bullet \\ P_N \end{bmatrix}$$



Lösungsrichtung

Zeile 1 $V_1 = \frac{1}{K_{1,1}} P_1$

Zeile 2 $V_2 = \frac{1}{K_{2,2}} \{P_2 - K_{2,1} V_1\}$

Zeile 3 $V_3 = \frac{1}{K_{3,3}} \{P_3 - K_{3,1} V_1 - K_{3,2} V_2\}$

Zeile j $V_j = \frac{1}{K_{j,j}} \left\{P_j - \sum_{k=1}^{j-1} K_{j,k} V_k\right\}$

Liegt eine *untere Dreiecksmatrix* vor (nur die Einträge unterhalb der Diagonalen sind ungleich Null), kann das Gleichungssystem auf triviale Art durch *Vorwärtseinsetzen* gelöst werden. Beginnend von oben wird eine Unbekannte nach der anderen auf der Basis der bereits errechneten bestimmt. Entsprechend wird ein System mit einer *oberen Dreiecksmatrix* durch *Rückwärtseinsetzen* gelöst.

Viele Gleichungslöser formen deshalb das Gleichungssystem so um, dass es durch Vorwärts- oder Rückwärtseinsetzen gelöst werden kann. Jedoch gibt es dort unterschiedliche Strategien.

Der Algorithmus nach GAUß

Das Eliminationsverfahren nach Gauß besteht darin, von oben beginnend *Vielfache von Gleichungen* so voneinander abzuziehen, dass die Koeffizientenmatrix zu einer oberen Dreiecksmatrix wird. Die Umformung von *Gleichungen* impliziert, dass die rechte Seite mit umgeformt wird. Hat man mehrere rechte Seiten (Lastfälle), werden diese *simultan* bearbeitet.

Ausgangsproblem

$$\begin{bmatrix} 10.00 & 4.00 & 2.00 \\ 4.00 & 20.00 & 1.00 \\ 2.00 & 1.00 & 5.00 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ V_3 \end{bmatrix} = \begin{bmatrix} 24.00 \\ 47.00 \\ 19.00 \end{bmatrix}$$

von Z2 abziehen: Z1*4/10

$$\begin{bmatrix} 10.00 & 4.00 & 2.00 \\ 0.00 & 18.40 & 0.20 \\ 2.00 & 1.00 & 5.00 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ V_3 \end{bmatrix} = \begin{bmatrix} 24.00 \\ 37.40 \\ 19.00 \end{bmatrix}$$

von Z3 abziehen: Z1*2/10

$$\begin{bmatrix} 10.00 & 4.00 & 2.00 \\ 0.00 & 18.40 & 0.20 \\ 0.00 & 0.20 & 4.60 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ V_3 \end{bmatrix} = \begin{bmatrix} 24.00 \\ 37.40 \\ 14.20 \end{bmatrix}$$

von Z3 abziehen: Z2*0.2/18.4

Dreiecksform: trivial lösbar

$$\begin{bmatrix} 10.00 & 4.00 & 2.00 \\ 0.00 & 18.40 & 0.20 \\ 0.00 & 0.00 & 4.5978 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ V_3 \end{bmatrix} = \begin{bmatrix} 24.00 \\ 37.40 \\ 13.7935 \end{bmatrix}$$

MENUM

Einige Gedanken zur Rechenzeit

$$\begin{bmatrix} K_{1,1} & K_{1,2} & K_{1,3} & \bullet\bullet\bullet & K_{1,N} \\ K_{2,1} & K_{2,2} & K_{2,3} & \bullet\bullet\bullet & K_{2,N} \\ K_{3,1} & K_{3,2} & K_{3,3} & \bullet\bullet\bullet & K_{3,N} \\ \bullet\bullet\bullet & \bullet\bullet\bullet & \bullet\bullet\bullet & \bullet\bullet\bullet & \bullet\bullet\bullet \\ K_{N,1} & K_{N,2} & K_{N,3} & \bullet\bullet\bullet & K_{N,N} \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ \bullet \\ V_N \end{bmatrix} = \begin{bmatrix} P_1 \\ P_2 \\ P_2 \\ \bullet \\ P_N \end{bmatrix}$$

Vollmatrix:

Bei der Elimination der Spalte 1 in der Matrix müssen N-1 Zeilen der Länge N verarbeitet werden, bei Spalte 2 N-2 Zeilen der Länge N-1 usw. Je weiter man sich in der Matrix nach rechts arbeitet, um so kürzer werden die Zeilen, da die vorne stehenden Nullen übersprungen werden können. Die rechte Seite besitzt aber immer nur einen einzigen Eintrag. Somit überwiegt die Anzahl der Rechenoperationen zur Umformung der Matrix diejenige zur simultanen Umformung der rechten Seite um ein Vielfaches.

Gebänderte Struktur:

Bei einer Bandstruktur entfallen alle Operationen außerhalb des Bandes. Auch wenn sich dadurch die Rechenzeit, abhängig von der Bandbreite, deutlich reduziert, wird dennoch der Löwenanteil der Rechenzeit für die Matrizenumformung verwendet.

Effizienzgrenzen des GAUß-Verfahrens: Lineare Zeitverlaufsberechnung mit dem CAA-Algorithmus

Initialisierung:
$\mathrm{I}: \ddot{\mathbf{V}}_0$

Initialisierung:
$\mathrm{II}: \mathbf{K}_{\mathrm{eff}}$

bekannt:
$\mathbf{V}_0, \dot{\mathbf{V}}_0, \ddot{\mathbf{V}}_0$

Berechnen:
$\mathbf{P}_{\mathrm{eff}\,1}, \mathbf{V}_1, \dot{\mathbf{V}}_1, \ddot{\mathbf{V}}_1$

bekannt:
$\mathbf{V}_1, \dot{\mathbf{V}}_1, \ddot{\mathbf{V}}_1$

Berechnen:
$\mathbf{P}_{\mathrm{eff}\,2}, \mathbf{V}_2, \dot{\mathbf{V}}_2, \ddot{\mathbf{V}}_2$

bekannt:
$\mathbf{V}_2, \dot{\mathbf{V}}_2, \ddot{\mathbf{V}}_2$

dynamisches Gleichgewicht:

$$\mathrm{K}_{\mathrm{eff}} \cdot \mathrm{V}_{\mathrm{k}} = \mathrm{P}_{\mathrm{eff,k}}$$

effektive Steifigkeitsmatrix (konstant):

$$\mathbf{K}_{\mathrm{eff}} = \mathbf{K} + \frac{4}{\Delta t^2}\mathbf{M} + \frac{2}{\Delta t}\mathbf{C}$$

effektiver Lastvektor (zeitveränderlich):

$$\mathbf{P}_{\mathrm{eff,k}} = \mathbf{P}_{\mathrm{k}} + \left\{\frac{4}{\Delta t^2}\mathbf{M} + \frac{2}{\Delta t}\mathbf{C}\right\}\mathbf{V}_{\mathrm{k-1}} + \left\{\frac{4}{\Delta t}\mathbf{M} + \mathbf{C}\right\}\dot{\mathbf{V}}_{\mathrm{k-1}} + \mathbf{M}\,\ddot{\mathbf{V}}_{\mathrm{k-1}}$$

Überblick

Bei einer linearen Zeitverlaufsberechnung (Beispiel hier: CAA-Algorithmus) wird die DGL in eine *Abfolge von linearen Gleichungssystemen* umgewandelt. Die Koeffizientenmatrix ist zeitlich konstant. Die rechten Seiten sind jedoch in jedem Zeitschritt anders, wobei sie auf den Ergebnissen des letzten Schrittes basieren. Somit können die rechten Seiten *nicht simultan* mit der Matrix umgeformt werden. Auch wenn das möglich wäre, würde die gleichzeitige Abspeicherung von mehreren tausend rechten Seiten einen in dieser Größe eventuell nicht mehr allokierbaren Speicherplatz verlangen.

Da bei der GAUß-Elimination die Verarbeitung der rechten Seite untrennbar mit der Umformung der Matrix verbunden ist, müsste die Matrixumformung in jedem Schritt erneut erfolgen. Das würde zunächst das Anlegen einer Kopie der Matrix verlangen, da diese bei der Gleichungslösung zerstört wird. Dann würden in jedem Zeitschritt zu 99 % oder mehr exakt die gleichen Rechenoperationen durchgeführt werden, da ja nur die rechte Seite anders ist als im Vorgängerschritt und die Matrix in jedem Schritt identisch ist.

Für derartige Probleme ist es wesentlich effizienter, eine Gleichungslösungstechnik zu verwenden, die die Umformung der Matrix von der Verarbeitung der rechten Seite trennt. Dann kann die Umformung der Matrix ein einziges Mal zu Beginn der Zeitschleife erfolgen. Ein derartiges Verfahren stellt die sog. *CHOLESKY-Faktorisierung* dar.

CHOLESKY-Faktorisierung - Grundidee

„Die Cholesky-Zerlegung (auch Cholesky-Faktorisierung) (nach *André-Louis Cholesky*, 1875–1918) bezeichnet in der numerischen Mathematik eine Zerlegung einer *symmetrischen, positiv definiten Matrix* in ein *Produkt* aus einer *unteren Dreiecksmatrix* und deren *Transponierter*. Sie wurde von Cholesky vor 1914 im Zuge der Triangulation Kretas durch den *service géographique de l'armée entwickelt*. Das Konzept kann auch allgemeiner für hermitesche Matrizen definiert werden.“ (Wikipedia, Hervorhebungen von W.Z.)

K wird als Produkt einer unteren Dreiecksmatrix L (lower triangular matrix) und einer oberen Dreicksmatrix U (upper triangular matrix) dargestellt. Ist K symmetrisch, gilt $U = L^T$.

$$\mathbf{K} = \mathbf{L} \cdot \mathbf{U} \qquad \mathbf{U} = \mathbf{L}^T$$

Das Gleichungssystem K·V=P wird damit zu L·U·V=P. Das Produkt U·V würde einen Vektor darstellen, den wir mit $\tilde{V}$ abkürzen. Das faktorisierte Gleichungssystem lässt sich trivial durch Vorwärts- und Rückwärtseinsetzen lösen.

$$\mathbf{K} \cdot \mathbf{V} = \mathbf{L} \cdot \mathbf{U} \cdot \mathbf{V} = \mathbf{P}$$

Substitution — **Lösung durch VE** — **Lösung durch RE**

$$\mathbf{U} \cdot \mathbf{V} = \tilde{\mathbf{V}} \;\rightarrow\; \mathbf{L} \cdot \tilde{\mathbf{V}} = \mathbf{P} \;\rightarrow\; \tilde{\mathbf{V}} \;\rightarrow\; \mathbf{U} \cdot \mathbf{V} = \tilde{\mathbf{V}} \;\rightarrow\; \mathbf{V}$$

MENUM

CHOLESKY-Faktorisierung - Algorithmus

$$\begin{bmatrix} L_{11} & 0 & 0 & \cdots & 0 \\ L_{21} & L_{22} & 0 & \cdots & 0 \\ L_{31} & L_{32} & L_{33} & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ L_{N1} & L_{N2} & L_{N3} & \cdots & L_{NN} \end{bmatrix} \cdot \begin{bmatrix} L_{11} & L_{21} & L_{31} & \cdots & L_{N1} \\ 0 & L_{22} & L_{32} & \cdots & L_{N2} \\ 0 & 0 & L_{33} & \cdots & L_{N3} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & L_{NN} \end{bmatrix} = \begin{bmatrix} K_{11} & K_{21} & K_{31} & \cdots & K_{N1} \\ K_{21} & K_{22} & K_{32} & \cdots & K_{N2} \\ K_{31} & K_{32} & K_{33} & \cdots & K_{N3} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ K_{N1} & K_{N2} & K_{N3} & \cdots & K_{NN} \end{bmatrix}$$

Zeile 1 von L mal U liefert die Spalte 1 von L

Z1xS1: $L_{11}L_{11} = K_{11} \rightarrow L_{11} = \sqrt{K_{11}}$

Z1xSj: $L_{11}L_{j1} = K_{j1} \rightarrow L_{j1} = \dfrac{K_{j1}}{L_{11}}$

Zeile 2 von L mal U liefert die Spalte 2 von L

Z2xS2: $L_{21}^2 + L_{22}^2 = K_{22} \rightarrow L_{22} = \sqrt{K_{22} - L_{21}^2}$

Z2xSj: $L_{21}L_{j1} + L_{22}L_{j2} = K_{j1} \rightarrow L_{j2} = \dfrac{K_{j1} - L_{21}L_{j1}}{L_{22}}$

Zeile k von L mal U liefert die Spalte k von L

ZkxSk: $L_{kk} = \sqrt{K_{kk} - \sum_{i=1}^{k-1} L_{ki}^2}$

ZkxSj: $L_{jk} = \dfrac{K_{kj} - \sum_{i=1}^{k-1} L_{ki} L_{ji}}{L_{kk}}$

Eigenschaften der CHOLESKY -Zerlegung

Speicherstruktur:

Die Skyline- oder Bandstruktur bleibt erhalten: Es werden keine Ungleichnullelemente außerhalb der Skyline oder des Bandes erzeugt. Eine Sparsestruktur wird hingegen zerstört, da innerhalb der Säulen Einträge von Null zu Nichtnull werden. Im ungünstigsten Fall würde die Faktorisierte zu einem Skylineschema werden.

Implementierung bei Skyline- oder Bandstruktur:

Die Implementierung kann „*in place*“ geschehen. Man kann den Algorithmus so implementieren, dass bis auf einige Hilfsvariablen kein neuer Speicherplatz notwendig wird, da die Ursprungsmatrix mit der Faktorisierten überschrieben wird.

Matrixeigenschaften:

Die Matrix muss *positiv definit* sein, da ansonsten mindestens ein Wurzelargument bei der Berechnung der Diagonalelemente Null oder negativ würde. Ein Nullargument wäre gleichbedeutend mit einer *singulären Matrix*. In diesem Fall wäre das Tragwerk *kinematisch verschieblich* und kein Gleichgewicht möglich. Im Fall negativer Wurzelargumente ist Gleichgewicht möglich, aber es würde sich um einen *instabilen Gleichgewichtszustand* handeln. Bei linearen Problemen ist das nicht möglich, bei nichtlinearen jedoch sehr wohl. Deshalb wäre die ursprüngliche CHOLESKY-Faktorisierung für nichtlineare Probleme nur eingeschränkt nutzbar.

LDL^T-Faktorisierung

Bei der LDL^T-Faktorierung werden die Diagonalglieder L_{jj} in eine eigene Diagonalmatrix D extrahiert und die Diagonaleinträge L_{jj} zu 1 gesetzt. Dadurch entfallen die Produkte der Diagonalglieder mit sich selbst und die Wurzel verschwindet. Jetzt können beliebige nichtsinguläre Matrizen faktorisiert werden.

$$\begin{bmatrix} 1 & 0 & 0 & \bullet\bullet & 0 \\ L_{21} & 1 & 0 & \bullet\bullet & 0 \\ L_{31} & L_{32} & 1 & \bullet\bullet & 0 \\ \bullet\bullet & \bullet\bullet & \bullet\bullet & \bullet\bullet & \bullet\bullet \\ L_{N1} & L_{N2} & L_{N3} & \bullet\bullet & 1 \end{bmatrix} \cdot \begin{bmatrix} D_{11} & 0 & 0 & \bullet\bullet & 0 \\ 0 & D_{22} & 0 & \bullet\bullet & 0 \\ 0 & 0 & D_{33} & \bullet\bullet & 0 \\ \bullet\bullet & \bullet\bullet & \bullet\bullet & \bullet\bullet & \bullet\bullet \\ 0 & 0 & 0 & \bullet\bullet & D_{NN} \end{bmatrix} \begin{bmatrix} 1 & L_{21} & L_{31} & \bullet\bullet & L_{N1} \\ 0 & 1 & L_{32} & \bullet\bullet & L_{N2} \\ 0 & 0 & 1 & \bullet\bullet & L_{N3} \\ \bullet\bullet & \bullet\bullet & \bullet\bullet & \bullet\bullet & \bullet\bullet \\ 0 & 0 & 0 & \bullet\bullet & 1 \end{bmatrix} = \begin{bmatrix} K_{11} & K_{21} & K_{31} & \bullet\bullet & K_{N1} \\ K_{21} & K_{22} & K_{32} & \bullet\bullet & K_{N2} \\ K_{31} & K_{32} & K_{33} & \bullet\bullet & K_{N3} \\ \bullet\bullet & \bullet\bullet & \bullet\bullet & \bullet\bullet & \bullet\bullet \\ K_{N1} & K_{N2} & K_{N3} & \bullet\bullet & K_{NN} \end{bmatrix}$$

Durch Ausmultiplizieren erhält man die entsprechenden Formeln für die Faktorisierung. Die Eigenschaften hinsichtlich Speicherstruktur und Implementierung gelten unverändert.

Diagonalglieder

$$D_{jj} = K_{jj} - \sum_{k=1}^{j-1} D_{kk} \cdot L_{jk}^2$$

Nichtdiagonalglieder

$$L_{ij} = \frac{1}{D_{jj}} \left\{ K_{ij} - \sum_{k=1}^{j-1} D_{kk} L_{jk} L_{ik} \right\}$$

Gleichungslösung mit der LDL^T-Faktorisierung

Gleichungssystem

$$\mathbf{K} \cdot \mathbf{V} = \mathbf{L} \cdot \mathbf{D} \cdot \mathbf{L}^T \cdot \mathbf{V} = \mathbf{P}$$

Substitution

$$\mathbf{D} \cdot \mathbf{L}^T \cdot \mathbf{V} = \tilde{\mathbf{V}}$$

Lösung durch VE

$$\mathbf{L} \cdot \tilde{\mathbf{V}} = \mathbf{P} \quad \rightarrow \quad \tilde{\mathbf{V}} \quad \rightarrow \quad \mathbf{D} \cdot \mathbf{L}^T \cdot \mathbf{V} = \tilde{\mathbf{V}}$$

Lösung durch RE

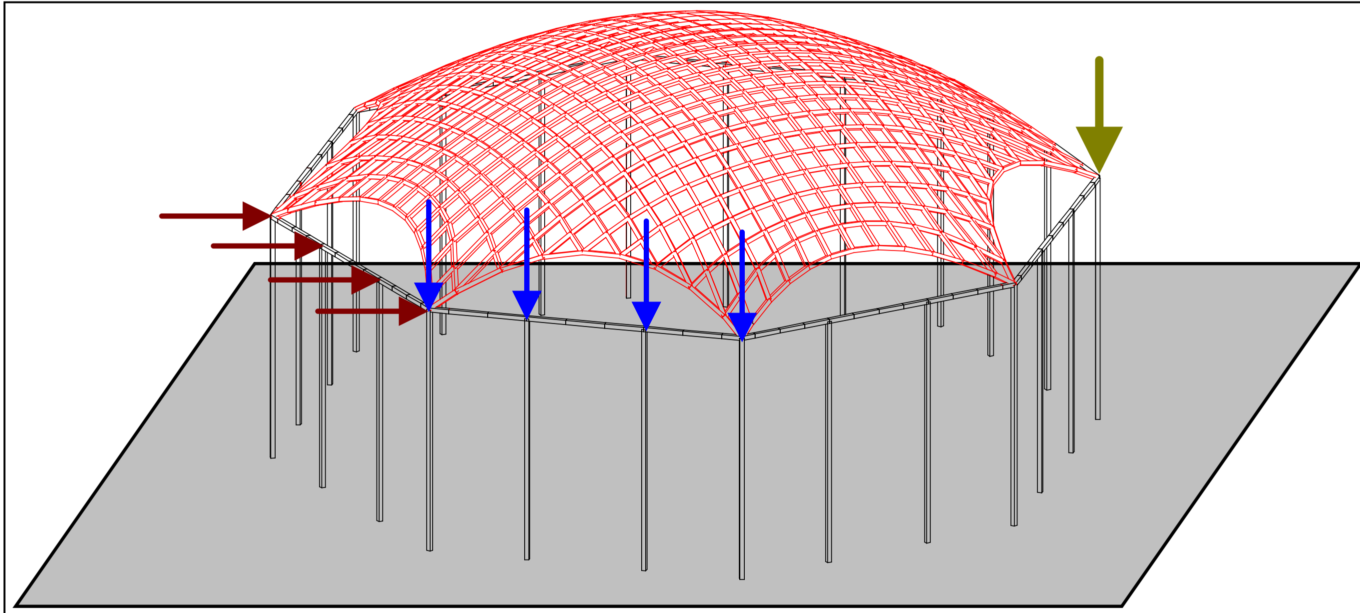
$$\mathbf{L}^T \cdot \mathbf{V} = \mathbf{D}^{-1} \cdot \tilde{\mathbf{V}} = \tilde{\mathbf{P}} \quad \rightarrow \quad \mathbf{V}$$

$$\tilde{P}_k = \frac{\tilde{V}_k}{D_{kk}}$$

Die Berechnung von Pschlange ist trivial, da das Produkt mit der Inversen D^{-1} nicht als Matrizenprodukt ausgeführt wird, sondern infolge der Diagonalstruktur von D einfach in der Division durch die Diagonalelemente D_{kk} besteht.

Pfadverfolgung

Tragwerk & Belastung



Das Tragwerk stellt eine beliebig komplexe Struktur aus verschiedenartigen Bauteiltypen dar, die mittels finiter Elemente diskretisiert wurde. Es wird durch eine Reihe von *Lastkollektiven* belastet.

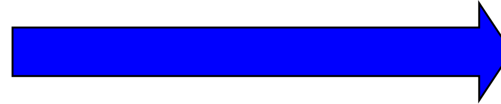
Lastkollektive:

- **Volumenlasten, Flächenlasten, Linienlasten**
- **Einzellasten**
- **Temperaturbeanspruchungen**
- **Stützensenkungen**
- **....**

menum

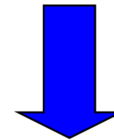
Lastgeschichten

Lastkollektive



m Lastfälle

$Q_1, Q_2, ..., Q_m$

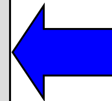


n Lastfallkombinationen:

$$P_1 = F_{11}\, Q_1 + ... + F_{1m}\, Q_m$$

...

$$P_n = F_{n1}\, Q_1 + ... + F_{nm}\, Q_m$$



Lastgeschichten

$$P = \lambda_1\, P_1 + \lambda_2\, P_2 + ...\, \lambda_n\, P_n$$

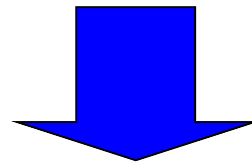
MENUM

Einparametrige Lastgeschichten

Jeder Lastfall bzw. Lastvektor ist mit einem Lastfaktor λ behaftet. Zu Beginn werden alle Lastfaktoren zu Null initialisiert – damit ist der konstante Lastanteil P_{konst} Null. Dann wird in einer ersten inkrementell-iterativen Rechnung ein Lastfall k ausgewählt und über seinen Lastfaktor λ_k beliebig gesteigert. Aufbauend darauf erfolgt eine zweite Rechnung mit Lastfall j, die jetzt infolge des eingefrorenen Lastfaktors λ_k eine konstante Vorbelastung aufweist. Auf diese Weise können beliebige Lastgeschichten nachgefahren werden, wobei stets *ein Lastfaktor aktiv* ist und die restlichen auf ihrem *zuletzt erreichten Niveau eingefroren* sind. Der Einfachheit halber wird der aktive Lastfaktor mit λ ohne Index bezeichnet.

Einparametrige Lastgeschichte:

- **ein Lastfaktor ist aktiv**
- **alle anderen sind eingefroren**



$$\mathbf{P}(\lambda) = \lambda \cdot \mathbf{P}_0 + \mathbf{P}_{konst}$$

$$\mathbf{P}_{konst} = \bar{\lambda}_1 \cdot \mathbf{P}_1 + \bar{\lambda}_2 \cdot \mathbf{P}_2 + \cdots + \bar{\lambda}_n \cdot \mathbf{P}_n$$

Pfadverfolgung

Lastfaktor λ

Maximum

Primärpfad

Umkehrpunkte

Minimum

Verzweigungspunkt

Sekundärpfad

Verzweigungspunkt

Tertiärpfad

Verformung V

Ziel:

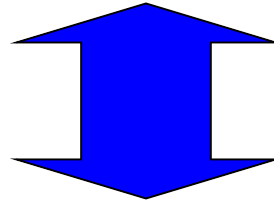
Die Verfolgung beliebiger Last-Verformungskurven unter Überwindung aller „schwierigen“ Punkte.

MENUM

Pfadverfolgungsalgorithmen

Klassen von Algorithmen:

- NEWTON/RAPHSON-Methoden
- Quasi-NEWTON-Methoden
- Bogenlängenverfahren



Kriterien für Algorithmen:

- Genauigkeit
- Robustheit
- Geschwindigkeit

MENUM

Konvergenz

Ein Computerprogramm benötigt ein zahlenmäßig definiertes Kriterium zur Entscheidung, ob Konvergenz erreicht wurde oder nicht – eine verbale Beschreibung wie „*Konvergenz liegt vor, wenn sich die Ergebnisse nicht mehr oder nur unwensentlich ändern*“ lässt sich nicht numerisch umsetzen. Da die Frage nach Konvergenz nicht einzelne Freiheitsgrade betrifft, sondern das Tragwerk als Ganzes, werden skalare Größen benötigt, die die Information eines *Systemvektors* mit N Freiheitsgraden in eine *einzige Zahl* abbildet.

Konvergenzkriterium:

$$\varepsilon_{cmp} = \frac{Z_{it}}{Z_{ref}} \leq \varepsilon_{tol}$$

***skalare* Variablen:**

Z_{it}: ***iterative* Norm**

Z_{ref}: ***Referenz*norm**

mittels des Programms berechnet ←

ε_{cmp}: **berechnete Genauigkeit**

ε_{tol}: **gewünschte Genauigkeit**

→ **durch den Benutzer festgelegt**

MENUM

Genauigkeitskriterien I: Vektornormen

EUKLIDISCHE Vektornorm:

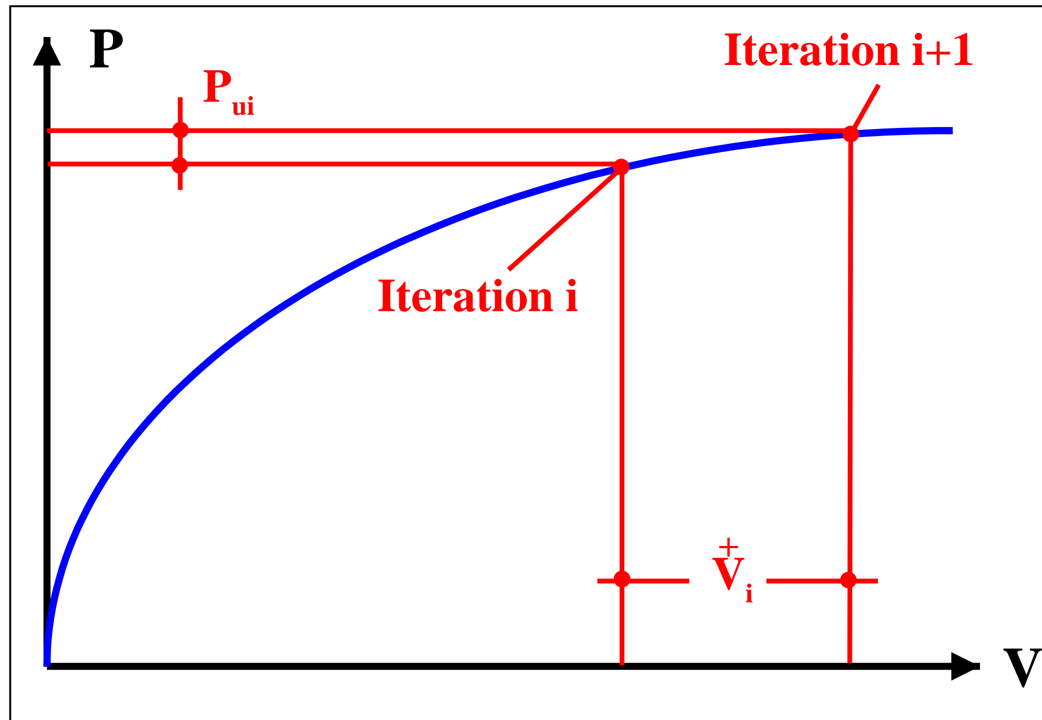
$$Z = \left\| \mathbf{V} \right\|_2 = \sqrt{\sum_{i=1}^{N} \left(V_i \right)^2}$$

V ist ein Verformungsvektor:
Verformungsnormen

V ist ein Kraftvektor:
Kraftnormen

- **Vektornormen besitzen *keine metrischen Eigenschaften*.**
- **Kraft- und Verschiebungsnormen wirken sich unterschiedlich auf die Fehler der Systemvariablen aus**

Aufweichende Struktur

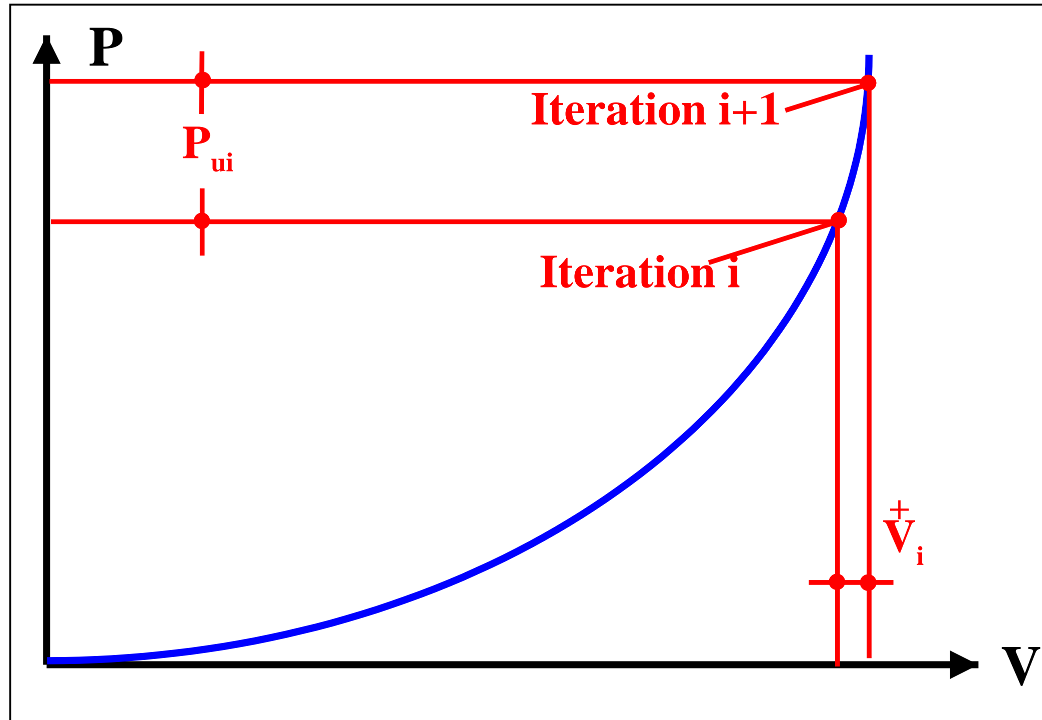


kleiner Fehler im Gleichgewicht eventuell großer Fehler bei den Verschiebungen

Verformungskriterium besser

MENUM

Versteifende Struktur



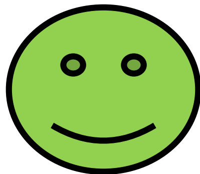
kleiner Fehler bei den Verschiebungen eventuell großer Fehler im Gleichgewicht

Kraftkriterium besser

Genauigkeitskriterien II: Energienormen

Energienorm: Kraftvektor x korrespondierender Verformungsvektor

$$Z = \mathbf{P}^{T} \cdot \mathbf{V}$$



- **Besitzt metrische Eigenschaften.**
- **Vereinigt in sich die Verformungs- und Kraftabfrage.**

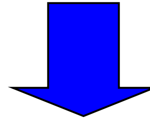
Referenznorm I

Möglichkeit 1:

Die Referenznorm bezieht sich auf die *Gesamtverschiebung* bzw. die *Gesamtlast*.

Beispiel: Energienorm

$$Z_{ref} = \mathbf{P}^{T}_{gesamt} \mathbf{V}_{gesamt} = \lambda \cdot \mathbf{P}^{T}_{0} \mathbf{V}_{gesamt}$$



Der zulässige *absolute Fehler* hängt von der Gesamtverschiebung ab und wächst mit wachsenden Lastfaktor λ.

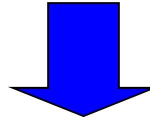
Referenznorm II

Möglichkeit 2:

Die Referenznorm bezieht sich auf die *inkrementelle Verschiebung des Lastschritts* bzw. das *Lastinkrement*.

Beispiel: Energienorm

$$Z_{ref} = \Delta\mathbf{P}^{T}\Delta\mathbf{V}_{0} = \Delta\lambda \cdot \mathbf{P}_{0}^{T}\Delta\mathbf{V}_{0}$$



Der zulässige *absolute Fehler* hängt von der *Größe des Lastschritts* ab, aber nicht mehr von der Gesamtverschiebung.

MENUM

Anwendung in der Praxis

In den unterschiedlichen Softwarepaketen sind unterschiedliche, oftmals mehrere Konvergenzabfragen implementiert. Der Benutzer kann diese auswählen und mit jeweiligen Genauigkeitsschranken versehen. Dabei kann die Option bestehen, mehrere Kriterien gleichzeitig zu aktivieren: Konvergenz ist dann erreicht, wenn sowohl das Verformungskriterium mit Genauigkeit $\varepsilon_{tol}=10^{-3}$ als das Kraftkriterium mit $\varepsilon_{tol}=10^{-4}$ erfüllt sind.

Die Wahl geeigneter Toleranzschranken erfordert eine gewisse Erfahrung, einen guten Kompromiss zwischen Genauigkeit und Rechenzeit zu finden. Auch bei heutigen Rechnerleistungen kann eine sehr kleine Genauigkeitsschranke zu extremen Rechenzeiten führen.

KEINE KONVERGENZ?

Wir haben bislang stillschweigend vorausgesetzt, dass sich Konvergenz einstellt. Das muss aber nicht sein: die Lösung kann *divergieren* oder *oszillieren* (hierzu später numerische Beispiele). Um zu verhindern, dass der Iterationszyklus bis in alle Ewigkeit weiterrechnet, definiert der Benutzer eine *maximal zulässige Iterationszahl*, nach deren Erreichen die Iteration terminiert. In einem Postmortem muss dann festgestellt werden, warum sich keine Konvergenz eingestellt hat.

NEWTON-RAPHSON-Verfahren

Das Standard-NEWTON/RAPHSON-Verfahren: Visualisierung

iterative Zwischenkonfigurationen
(nicht im Gleichgewicht)

λ

$\Delta\lambda$

Nachbarzustand
(im Gleichgewicht)

Grundzustand
(im Gleichgewicht)

V

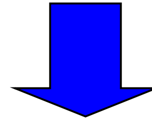
$\overset{+}{V}_0$ $\overset{+}{V}_1$ $\overset{+}{V}_2$ $\overset{+}{V}_3$ $\overset{+}{V}_4$

Die Verschiebungsinkremente werden mit der jeweils *exakten tangentialen Steifigkeit* berechnet, die zu Beginn jedes Lastschritts und jeder Iteration neu berechnet wird.

Standard NEWTON/RAPHSON: Eigenschaften

Rein *kraftgesteuerte* Berechnung:

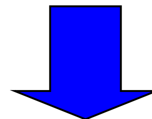
Der Algorithmus versucht, Gleichgewicht für ein *fixes vorgegebenes Lastniveau* zu finden



☺ **Bemessungszwecke: Lastniveau kann genau kontrolliert werden.**

☹ **Pfadverfolgung: ein Maximum kann nicht überwunden werden.**

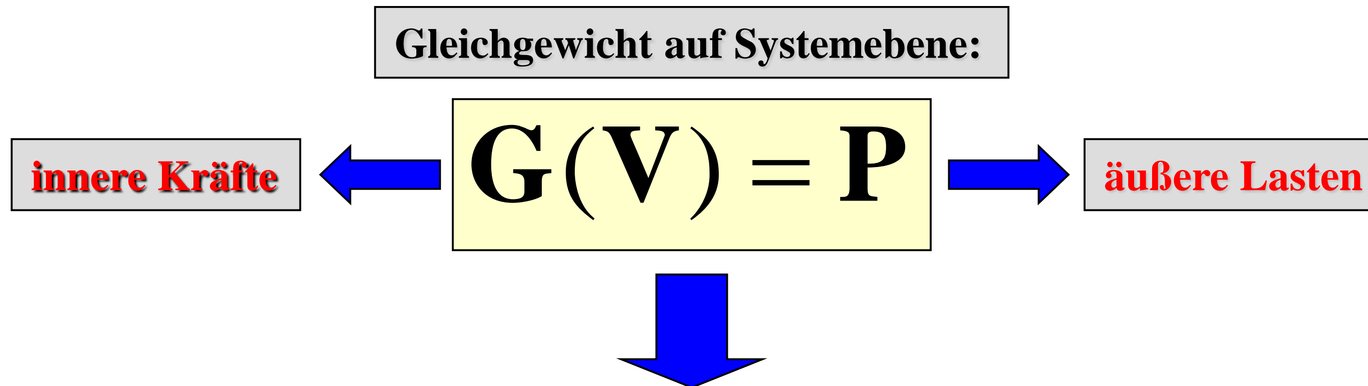
In der Nähe der Lösung weist der Algorithmus *quadratische Konvergenz* auf, d.h. der Fehler nimmt *quadratisch* ab: $\varepsilon = 10^{-1}, 10^{-2}, 10^{-4}, 10^{-8}$.



☺ **Schnelle Konvergenz: es werden wenige Iterationen benötigt.**

☹ **Jede Iteration ist wegen des Neuaufbaus der tangentialen Matrix aufwändig.**

Modifizierter NEWTON/RAPHSON: Grundidee



Wenn Gleichgewicht herrscht, herrscht Gleichgewicht! **Wie man den Gleichgewichtszustand findet, ist im Prinzip egal – man könnte diesen auch durch Erraten finden (Achtung: Diese Aussage trifft nur bei *pfadunabhängigen* Problemen zu!). Somit ist es nicht unbedingt notwendig, die exakte tangentiale Matrix während der Iteration zu benutzen – man kann auch mehr oder weniger ungenaue Matrizen verwenden. Theoretisch könnte man auch mit einer Einheitsmatrix iterieren. Die Iterationszahl wäre dann aber unvertretbar hoch. Liegt die „falsche Matrix“ zu weit von der exakten Matrix entfernt, kann es auch zu Konvergenzproblemen kommen.**

Durchführung eines Lastschritts

Laststeigerung um ΔP

- Aufbau vom K_{T0}
- Faktorisierung von K_{T0}
- Gleichungslösung für ΔP: ⇨ ΔV_0
- Update: $V_{ges} = V_{ges} + \Delta V_0$

→ $j = 1$ →

Iterationszyklus

- Aufbau vom K_{Tj}
- Aufbau von F_i
- Faktorisierung von K_{Tj}
- Gleichungslösung für P_{uj}: ⇨ ΔV_j
- Update: $V_{ges} = V_{ges} + \Delta V_j$

↓

Konvergenz?

ja: nächster Lastschritt

nein: neue Iteration

$j = j+1$

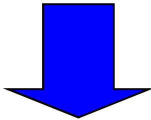
rot:	rechenzeitaufwändig
dunkelrot:	mittelaufwändig
grün:	rechenzeitunaufwändig

MENUM

Modifizierter NEWTON/RAPHSON: Eigenschaften

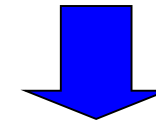
Ablauf des Verfahrens:

- **Nur zu Beginn jedes Lastschritts wird die tangentiale Matrix aufgebaut und faktorisiert.**
- **In den Iterationen wird die bereits faktorisierte Matrix der Laststeigerung zur Gleichungslösung verwendet**



- **Jede einzelne Iteration läuft schneller.**
- **Quadratische Konvergenz geht verloren: es werden mehr Iterationen benötigt.**
- **Der Algorithmus verliert eventuell an Robustheit.**

Die Gesamtrechenzeit kann sinken, aber auch wachsen



Numerische Beispiele

Modifizierter NEWTON/RAPHSON: Visualisierung

iterative Zwischenkonfigurationen
(nicht im Gleichgewicht)

λ

$\Delta\lambda$

Nachbarzustand
(im Gleichgewicht)

Grundzustand
(im Gleichgewicht)

V

$\overset{+}{V}_0$ $\overset{+}{V}_1$ $\overset{+}{V}_2$ $\overset{+}{V}_3$

MENUM

Quasi-NEWTON-Verfahren

Quasi-Newton-Methoden

Das modifizierte NEWTON/RAPHSON-Verfahren ist durch die Tatsache motiviert, dass man nicht unbedingt die exakte tangentiale Matrix verwenden muss, um sich iterativ an die wahre Lösung heranzuarbeiten. Hierbei wurde die einfachste, aber auch schlechteste Variante gewählt: man übernimmt die Matrix des Laststeigerungsschritts und verändert diese während der Iteration überhaupt nicht. Dies spart ein Maximum an Rechenzeit, führt bei starker Nichtlinearität jedoch zu einer großen Diskrepanz zwischen der wahren tangentialen Matrix, die optimale Konvergenz liefert, und der verwendeten Matrix. Als Folge steigt die Iterationszahl stark an, besonders wenn hohe Genauigkeiten erreicht werden sollen.

Somit stellt sich die Frage, ob man nicht beides haben kann: ein gutes Konvergenzverhalten bei gleichzeitig reduzierter Rechenzeit während der Iterationen. Dieser Wunsch führt auf sog. *Update-Verfahren*. Hierbei wird keine neue Matrix mittels Durchlauf durch die finiten Elemente berechnet, sondern es wird die neue Matrix durch ein Update der alten Matrix erzeugt. Dadurch verbessert sich die Matrix während der Iterationen, welches zu einer Reduktion der Iterationszahl führt. Die Verbesserung der Matrix ist jedoch mit erheblich weniger Rechenzeit verbunden als zum Aufbau der echten tangentialen Matrix erforderlich ist.

Im eindimensionalen Fall erreicht man dieses Ziel durch ein echtes *Sekantenverfahren*. Bei Systemen mit mehr Freiheitsgraden spricht man von *Quasi-Newton-Methoden*.

Sekantenverfahren bei einem Freiheitsgrad: Visualisierung

P

$K_{T3}=K_{S2}$

$K_{T1}=K_{S0}$

$K_{T2}=K_{S1}$

K_{S2}

K_{T0}

K_{S1}

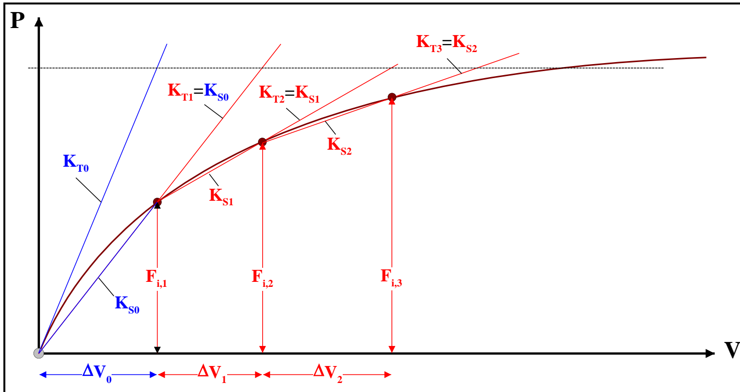
$F_{i,1}$ $F_{i,2}$ $F_{i,3}$

K_{S0}

V

ΔV_0 ΔV_1 ΔV_2

Sekantenverfahren bei einem Freiheitsgrad: Ableitung der Sekantenmatrix



Matrix des Iterationsschrittes j+1

$$K_{T,j+1} = K_{S,j} = \frac{F_{i,j+1} - F_{i,j}}{\Delta V_j}$$

Die Sekantenmatrix liegt zwischen der alten und neuen Tangente. Sie konvergiert im Verlauf der Iteration gegen die wahre Tangente.

Tangentengleichung bei Iteration 0 (Laststeigerung)

$$K_{T0} \cdot \Delta V_0 = \Delta P$$

$$\Delta V_0$$

$$V_1 = V_0 + \Delta V_0$$

$$F_{i,1} = F_i(V_1)$$

Sekantengleichung bei Iteration 0

$$K_{S,0} \cdot \Delta V_0 = F_{i,1} - F_{i,0}$$

$$K_{S,0} = \frac{F_{i,1} - F_{i,0}}{\Delta V_0}$$

$$K_{T,1} = K_{S,0}$$

Sekantenverfahren bei zwei Freiheitsgraden?

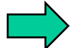
Tangentengleichung bei Iteration 0

$$\begin{bmatrix} K_{T11} & K_{T12} \\ K_{T21} & K_{T22} \end{bmatrix} \begin{bmatrix} \Delta V_{1,0} \\ \Delta V_{2,0} \end{bmatrix} = \begin{bmatrix} \Delta P_1 \\ \Delta P_2 \end{bmatrix}$$

$$\begin{bmatrix} \Delta V_{1,0} \\ \Delta V_{2,0} \end{bmatrix}$$

$$\begin{bmatrix} V_{1,1} \\ V_{2,1} \end{bmatrix} = \begin{bmatrix} V_{1,0} \\ V_{2,0} \end{bmatrix} + \begin{bmatrix} \Delta V_{1,0} \\ \Delta V_{2,0} \end{bmatrix}$$

$$\begin{bmatrix} F_{i1,1} \\ F_{i2,1} \end{bmatrix}$$



Sekantengleichung bei Iteration 0

$$\begin{bmatrix} K_{S11} & K_{S12} \\ K_{S21} & K_{S22} \end{bmatrix} \begin{bmatrix} \Delta V_{1,0} \\ \Delta V_{2,0} \end{bmatrix} = \begin{bmatrix} F_{i1,1} \\ F_{i2,1} \end{bmatrix} - \begin{bmatrix} F_{i1,0} \\ F_{i2,0} \end{bmatrix}$$

Bei *zwei Freiheitsgraden* besteht die Sekantengleichung aus *zwei Gleichungen*. Die dort auftauchende Sekantenmatrix enthält jedoch bei Ausnutzung ihrer Symmetrie *drei Einträge*. Damit ist das Problem unterbestimmt, und es ist nicht möglich, eine eindeutige Sekantenmatrix zu bestimmen. Es gibt unendlich viele Matrizen, die die Sekantengleichung erfüllen.

Man muss folglich Annahmen hinsichtlich des Aufbaus der Sekantenmatrix treffen. Diese Annahmen sehen vor, die Sekantenmatrix durch ein Update aus der alten Matrix zu gewinnen. Das Update muss so gestaltet sein, dass die Sekantengleichung erfüllt wird.

MENUM

Matrizenupdates I

Eine Matrix kann dadurch aktualisiert werden, dass auf die alte Matrix eine Summe von Produkten aus jeweils einem Zeilenvektor a mit einem Spaltenvektor b^T addiert wird.

$$\mathbf{K}_{neu} = \mathbf{K}_{alt} + \mathbf{a}_1 \cdot \mathbf{b}_1^T + \mathbf{a}_2 \cdot \mathbf{b}_2^T + \ldots + \mathbf{a}_n \cdot \mathbf{b}_n^T$$

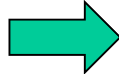
Jeder Summand j in dem Gesamtupdate liefert eine Vollmatrix ΔK_j. Diese Vollmatrix besitzt jedoch nur den *Rang 1*, d.h. nur die erste Zeile ist unabhängig. Die restlichen Zeilen ergeben sich unmittelbar als Vielfache der ersten Zeile.

$$\Delta\mathbf{K}_j = \mathbf{a}_j \cdot \mathbf{b}_j^T$$

Beispiel für die Rank-1-Eigenschaft

$$\mathbf{a} = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}^T$$

$$\mathbf{b} = \begin{bmatrix} 4 & 5 & 6 \end{bmatrix}^T$$



$$\Delta\mathbf{K} = \mathbf{a} \cdot \mathbf{b}^T = \begin{bmatrix} 4 & 5 & 6 \\ 8 & 10 & 12 \\ 12 & 15 & 18 \end{bmatrix}$$

Matrizenupdates II

Gesamtupdate als Summe von Einzelupdates:

$$\mathbf{K}_{neu} = \mathbf{K}_{alt} + \Delta\mathbf{K}_1 + \Delta\mathbf{K}_2 + \ldots + \Delta\mathbf{K}_n$$

$$\Delta\mathbf{K}_j = \mathbf{a}_j \cdot \mathbf{b}_j^T$$

Das Gesamtupdate $\Delta K = \Delta K_1 + \Delta K_2 + \ldots + \Delta K_n$ besitzt also den Rang n. Je nach Wahl von n spricht man dann von Rang1-Updates, Rang2-Updates usw. Die entscheidende Frage lautet jetzt: wie wählt man die Vektoren a und b? Es muss gelten: die durch das Update gewonnene Matrix muss die Sekantengleichung, auch *Quasi-Newton-Gleichung* genannt, erfüllen.

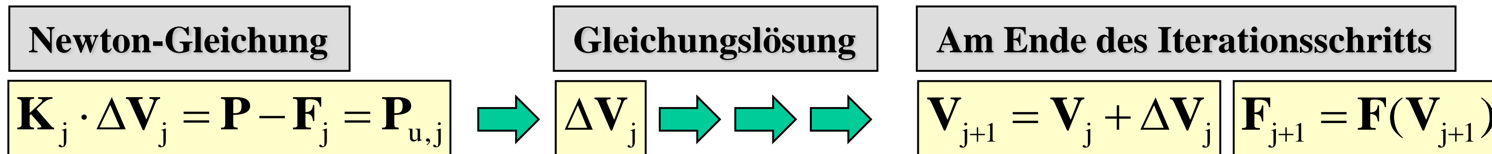
Die nachfolgende Herleitung lehnt sich an die Darstellung in „*Non-linear Finite Element Analysis of Solids and Structures*" von de Borst, Crisfield, Remmers, Verhoosel (Wiley & Sons 2012) an, wenngleich dort eine andere Notation verwendet wird.

Update: reguläre Iteration

Wir betrachten eine Iteration j (die Laststeigerung wäre dann die Iteration 0). Zu *Beginn* der Iteration sind bekannt:

Gesamtlast	Vektor der inneren Kräfte	Steifigkeitsmatrix	Gesamtverschiebung
$\mathbf{P}$	$\mathbf{F}_j$ $\mathbf{F}_0 = \mathbf{P}$	$\mathbf{K}_j$ $\mathbf{K}_0 = \mathbf{K}_{T0}$	$\mathbf{V}_j$

Mittels der Newton-Gleichung wird der Zuwachs innerhalb dieser Iteration berechnet. Im Falle der Iteration 0 liegt eine Laststeigerung vor und die rechte Seite wird zum Lastinkrement ΔP, ansonsten enthält die rechte Seite die Ungleichgewichtskräfte $P_{u,j}$.



Rang1-Update: Ansatz

Aus den Verschiebungen und den inneren Kräften am Anfang und am Ende der Iteration soll eine Sekantenmatrix erzeugt werden. Diese muss die Eigenschaft haben, die Sekantengleichung bzw. Quasi-Newton-Gleichung zu befriedigen.

Quasi-Newton-Gleichung

$$\mathbf{K}_{Sj} \cdot \Delta\mathbf{V}_j = \mathbf{F}_{j+1} - \mathbf{F}_j = \Delta\mathbf{F}_{j+1}$$

Das einfachste Update ist ein Rang1-Update. Die nachfolgende Update-Vorschrift erfüllt die Quasi-Newton-Gleichung. Sie enthält neben den bekannte Größen am Anfang und Ende des Iterationsschritts den zunächst beliebigen Vektor u, über den die Konvergenzeigenschaften optimiert werden können.

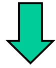
Rang1-Update

$$\mathbf{K}_{Sj} = \mathbf{K}_j + \frac{(\Delta\mathbf{F}_{j+1} - \mathbf{K}_j \Delta\mathbf{V}_j) \cdot \mathbf{u}_j^T}{\mathbf{u}_j^T \cdot \Delta\mathbf{V}_j}$$

Rang1-Update: Beweis

Einsetzen in die Pseudo-Newton-Gleichung

$$\left\{\mathbf{K}_j + \frac{(\Delta\mathbf{F}_{j+1} - \mathbf{K}_j\Delta\mathbf{V}_j)\cdot\mathbf{u}_j^T}{\mathbf{u}_j^T\cdot\Delta\mathbf{V}_j}\right\}\cdot\Delta\mathbf{V}_j = \Delta\mathbf{F}_{j+1}$$



Substitutionen

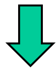
$$\mathbf{K}_j\cdot\Delta\mathbf{V}_j = \mathbf{P} - \mathbf{F}_j$$

$$\Delta\mathbf{F}_{j+1} = \mathbf{F}_{j+1} - \mathbf{F}_j$$

$$\mathbf{K}_j\cdot\Delta\mathbf{V}_j + \frac{(\mathbf{F}_{j+1} - \mathbf{F}_j - (\mathbf{P} - \mathbf{F}_j))\cdot\mathbf{u}_j^T}{\mathbf{u}_j^T\cdot\Delta\mathbf{V}_j}\cdot\Delta\mathbf{V}_j = \mathbf{F}_{j+1} - \mathbf{F}_j$$

$$\mathbf{K}_j\cdot\Delta\mathbf{V}_j + \mathbf{F}_{j+1} - \mathbf{P} = \mathbf{F}_{j+1} - \mathbf{F}_j$$

Das Produkt aus der Transponierten von u_j und ΔV_j liefert einen Skalar, der aus dem Bruch gekürzt werden kann.



Newton-Gleichung

$$\mathbf{K}_j\cdot\Delta\mathbf{V}_j = \mathbf{P} - \mathbf{F}_j$$

Die inneren Kräfte F_{j+1} heben sich auf und es ergibt sich die Newton-Gleichung. Sofern also der Verschiebungszuwachs ΔV_j aus der Newton-Gleichung ermittelt wurde, erfüllt das angegeben Update auch die Quasi-Newton-Gleichung und besitzt damit Sekanteneigenschaft.

Rang1-Update: BROYDEN-Update

Das Rang1-Update erfüllt die Quasi-Newton-Gleichung für alle Vektoren u. Die Wahl von u beeinflusst das Iterationsverhalten. Ein bekanntes Update ist das von *Charles George BROYDEN*, der u wie folgt wählt:

BROYDEN-Update

$$\mathbf{u}_j = \Delta \mathbf{V}_j$$

Damit ließe sich die Sekantenmatrix, welche dann als Approximation der Tangentenmatrix in der nächsten Iteration verwendet würde, ausschließlich aus bekannten Größen der aktuellen Iteration berechnen.

Das Update erfolgt durch rechenzeitunintensive Matrixmultiplikationen. Damit würde man zwar den Aufwand des Aufbaus der tangentialen Matrix wesentlich reduzieren, aber man müsste doch in jedem Schritt die aktualisierte Matrix neu faktorisieren.

Dies kann vermieden werden, indem man das Update nicht auf die Matrix, sondern direkt auf die Inverse bzw. Faktorisierte anwendet.

Update der Inversen

Ausgangpunkt bildet die ***SHERMAN-MORRISON-Gleichung*** (nach Jack SHERMAN und Winifred MORRISON), die einen Zusammenhang zwischen der Inversen eines Rang1-Updates und der Inversen der Ausgangsmatrix herstellt.

SHERMAN-MORRISON-Gleichung

$$\left\{\mathbf{A}+\mathbf{a}\cdot\mathbf{b}^T\right\}^{-1}=\mathbf{A}^{-1}-\frac{\mathbf{A}^{-1}\cdot\mathbf{a}\cdot\mathbf{b}^T\cdot\mathbf{A}^{-1}}{1+\mathbf{b}^T\cdot\mathbf{A}^{-1}\cdot\mathbf{a}}$$

Zuordnungen

$$\mathbf{A}=\mathbf{K}_j \qquad \mathbf{a}=\Delta\mathbf{F}_{j+1}-\mathbf{K}_j\Delta\mathbf{V}_j \qquad \mathbf{b}=\frac{\mathbf{u}_j^T}{\mathbf{u}_j^T\cdot\Delta\mathbf{V}_j}$$

einige Umformungen …

Rang1-Update der inversen Sekantenmatrix

$$\mathbf{K}_{Sj}^{-1}=\mathbf{K}_j^{-1}+\frac{(\Delta\mathbf{V}_j-\mathbf{K}_j^{-1}\Delta\mathbf{F}_{j+1})\cdot\mathbf{u}_j^T\cdot\mathbf{K}_j^{-1}}{\mathbf{u}_j^T\cdot\mathbf{K}_j^{-1}\cdot\Delta\mathbf{F}_{j+1}}$$

Damit entfällt der Faktorisierungsauswand in jeder Iteration. Allerdings hat diese Darstellung den Nachteil, dass durch das Update das Speicherimage zerstört würde und die aktualisierte Faktorisierte zur Vollmatrix würde, was zu unakzeptabel hohem Speicherplatzbedarf führt.

MENUM

Update des Ergebnisses

Zur Umgehung der Vollmatrix wird der Verschiebungszuwachs der Iteration j direkt durch den Verschiebungszuwachs der vorherigen Iteration j-1 ausgedrückt:

Newton-Gleichung der Iteration j

$$\mathbf{K}_j \cdot \Delta \mathbf{V}_j = \mathbf{P} - \mathbf{F}_j = \mathbf{P}_{u,j} \quad \Rightarrow \quad \Delta V_j = K_j^{-1} \cdot (P - F_j)$$

Die inverse Matrix der Iteration j entstand als Update der Matrix des Schrittes j-1

$$\mathbf{K}_j^{-1} = \mathbf{K}_{j-1}^{-1} + \frac{(\Delta \mathbf{V}_{j-1} - \mathbf{K}_{j-1}^{-1} \cdot \Delta \mathbf{F}_j) \cdot \mathbf{u}_{j-1}^T \cdot \mathbf{K}_{j-1}^{-1}}{\mathbf{u}_{j-1}^T \cdot \mathbf{K}_{j-1}^{-1} \cdot \Delta \mathbf{F}_j} = \left\{ \mathbf{I} + \frac{(\Delta \mathbf{V}_{j-1} - \mathbf{K}_{j-1}^{-1} \cdot \Delta \mathbf{F}_j) \cdot \mathbf{u}_{j-1}^T}{\mathbf{u}_{j-1}^T \cdot \mathbf{K}_{j-1}^{-1} \cdot \Delta \mathbf{F}_j} \right\} \cdot \mathbf{K}_{j-1}^{-1}$$

Einsetzen in die Newton-Gleichung der Iteration j

$$\Delta \mathbf{V}_j = \left\{ \mathbf{I} + \frac{(\Delta \mathbf{V}_{j-1} - \mathbf{K}_{j-1}^{-1} \cdot \Delta \mathbf{F}_j) \cdot \mathbf{u}_{j-1}^T}{\mathbf{u}_{j-1}^T \cdot \mathbf{K}_{j-1}^{-1} \cdot \Delta \mathbf{F}_j} \right\} \cdot \mathbf{K}_{j-1}^{-1} \cdot (\mathbf{P} - \mathbf{F}_j)$$

Abkürzung

$$\mathbf{w}_j = \mathbf{K}_{j-1}^{-1} \cdot (\mathbf{P} - \mathbf{F}_j) \Rightarrow \mathbf{w}_j = \mathbf{K}_{j-1}^{-1}(\mathbf{P} - \mathbf{F}_j + \mathbf{F}_{j-1} - \mathbf{F}_{j-1}) \Rightarrow \mathbf{w}_j = \mathbf{K}_{j-1}^{-1}(\mathbf{P} - \mathbf{F}_{j-1}) - \mathbf{K}_{j-1}^{-1}(\mathbf{F}_j - \mathbf{F}_{j-1})$$

$$\mathbf{w}_j = \Delta\mathbf{V}_{j-1} - \mathbf{K}_{j-1}^{-1}\Delta\mathbf{F}_j \Rightarrow \mathbf{K}_{j-1}^{-1}\Delta\mathbf{F}_j = \Delta\mathbf{V}_{j-1} - \mathbf{w}_j$$

Einsetzen in den Ausdruck für $\Delta\mathbf{V}_j$

$$\Delta\mathbf{V}_j = \left\{\mathbf{I} + \frac{(\Delta\mathbf{V}_{j-1} - (\Delta\mathbf{V}_{j-1} - \mathbf{w}_j)) \cdot \mathbf{u}_{j-1}^T}{\mathbf{u}_{j-1}^T \cdot (\Delta\mathbf{V}_{j-1} - \mathbf{w}_j)}\right\} \cdot \mathbf{w}_j \Rightarrow \Delta\mathbf{V}_j = \left\{\mathbf{I} + \frac{\mathbf{w}_j \cdot \mathbf{u}_{j-1}^T}{\mathbf{u}_{j-1}^T \cdot (\Delta\mathbf{V}_{j-1} - \mathbf{w}_j)}\right\} \cdot \mathbf{w}_j$$

Abkürzung

$$\alpha_j = (\mathbf{u}_{j-1}^T \cdot (\Delta\mathbf{V}_{j-1} - \mathbf{w}_j))^{-1} \Rightarrow \Delta\mathbf{V}_j = (\mathbf{I} + \alpha_j \mathbf{w}_j \mathbf{u}_{j-1}^T)\mathbf{w}_j \Rightarrow \Delta\mathbf{V}_j = \mathbf{w}_j + \alpha_j \mathbf{w}_j \mathbf{u}_{j-1}^T \mathbf{w}_j$$

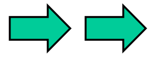
Abkürzung

$$\beta_j = \mathbf{u}_{j-1}^T \mathbf{w}_j \Rightarrow \Delta\mathbf{V}_j = \mathbf{w}_j + \alpha_j \beta_j \mathbf{w}_j$$

Update der Ergebnisse: BROYDEN-Update

BROYDEN-Update

$\mathbf{u}_j = \Delta\mathbf{V}_j$



$$\mathbf{w}_j = \mathbf{K}_{j-1}^{-1} \cdot (\mathbf{P} - \mathbf{F}_j)$$

$$\alpha_j = (\Delta\mathbf{V}_{j-1}^T \cdot (\Delta\mathbf{V}_{j-1} - \mathbf{w}_j))^{-1}$$

$$\beta_j = \Delta\mathbf{V}_{j-1}^T \mathbf{w}_j$$

$$\Delta\mathbf{V}_j = \mathbf{w}_j + \alpha_j \beta_j \mathbf{w}_j$$

Der Hilfsvekor w_j beinhaltet den Verschiebungszuwachs infolge der aktuellen Ungleichgewichtskräfte, ermittelt mit der Steifigkeit des vorherigen Schrittes.

Mit den Ungleichgewichtskräften des aktuellen Schrittes und dem Verschiebungsinkrement des vorherigen Schrittes lässt sich der Verschiebungszuwachs des aktuellen Schrittes berechnen. Der Verschiebungszuwachs des vorherigen Schrittes wiederum ergibt sich aus dem Verschiebungszuwachs des vorvorherigen Schrittes usw. bis zur Iteration 0. Somit lässt sich der aktuelle Verschiebungszuwachs rekursiv bis zum Verschiebungszuwachs des Lastinkrementes zurückverfolgen. Für diesen Urverschiebungszuwachs wird die tangentiale Matrix aufgebaut und faktorisiert. Ab dann erfolgt die Berechnung innerhalb des Iterationszyklus ausschließlich durch rekursive Updateoperationen. Die zu aktualisierende Größe ist der Hilfsvektor w.

Ablauf des Rang1-Updates

Iteration 0

$$\Delta\mathbf{V}_0 = \mathbf{K}_{T0}^{-1}\Delta\mathbf{P}$$

Tangentiale Matrix aufbauen, faktorisieren, Gleichungssystem lösen.

Iteration 1

$$\mathbf{w}_1 = \mathbf{K}_{T0}^{-1}\mathbf{P}_{u1}$$

Lösung des Gleichungssystems für die aktuellen Ungleichgewichtskräfte mit der Faktorisierten des Schritts 0.

Iteration 2

$$w_2 = K_{T1}^{-1}P_{u2}$$

Lösung des Gleichungssystems für die aktuellen Ungleichgewichtskräfte mit der Faktorisierten des Schritts 0 und Update mit den Ergebnissen der Iteration 1.

$$\mathbf{w}_2 = (\mathbf{I} + \alpha_1 w_1 \Delta\mathbf{V}_1^T)\mathbf{K}_{T0}^{-1}\mathbf{P}_{u2}$$

Iteration 3

$$\mathbf{w}_3 = (\mathbf{I} + \alpha_2 w_2 \Delta\mathbf{V}_2^T)(\mathbf{I} + \alpha_1 w_1 \Delta\mathbf{V}_1^T)\mathbf{K}_{T0}^{-1}\mathbf{P}_{u3}$$

Endgültige Formulierung

Rekursionsformel für w_j

$$\mathbf{w}_j = \prod_{k=1}^{j-1} (\mathbf{I} + \alpha_k \mathbf{w}_k \Delta\mathbf{V}_k^T) \cdot \mathbf{K}_{T0}^{-1} \mathbf{P}_{u,j}$$

Hilfswerte

$$\alpha_j = (\Delta\mathbf{V}_{j-1}^T \cdot (\Delta\mathbf{V}_{j-1} - \mathbf{w}_j))^{-1}$$

$$\beta_j = \Delta\mathbf{V}_{j-1}^T \mathbf{w}_j$$

Verschiebungsupdate

$$\Delta\mathbf{V}_j = \mathbf{w}_j + \alpha_j \beta_j \mathbf{w}_j$$

In jeder Iteration wird das Gleichungssystem für die aktuellen Ungleichgewichtskräfte mit der bereits faktorisierten Ausgangsmatrix gelöst, welches wenig Rechenzeit in Anspruch nimmt.

Die Rekursionsformel kann so implementiert werden, dass die formal in der Klammer entstehende Matrix infolge des Rang1-Charakters des Produkts zwischen w und ΔV nicht aufgebaut werden muss. Die Auswertung der Rekursionsformel stellt dann eine Abfolge von reinen Vektoroperationen dar.

Allerdings ist es notwendig, für den ganzen Iterationszyklus die Vektoren w und ΔV sowie die Faktoren α zu speichern. Das stellt allerdings bei „normalen“ Iterationszahlen von maximal 30 Iterationen kein Problem dar.

Zusammenfassung

Das hier vorgestellte BROYDEN-Update mit Rang 1 stellt nur ein mögliches Update dar. Es lassen sich weitere Rang1-Updates ableiten, und darüber hinaus auch Updates mit höherem Rang, die entsprechend komplexer formuliert sind.

Das bekannteste Rang2-Update stammt von Broyden-Fletcher–Goldfarb-Shanno und heißt nach den Initialen seiner Erfindern (Charles BROYDEN, Roger FLETCHER, Donald GOLDFARB, David SHANNO) BFGS-Update. Es findet sich in vielen FE-Programmsystemen, wird aber hier aus Platzmangel nicht weiter besprochen.

Abschließend zu den Update-Verfahren sei aus dem bereits erwähnten Buch von de Borst et al zitiert: „Although significant gains have been reported in terms of computer time, others report a somewhat erratic behaviour of Quasi-Newton methods with a non-monotonous convergence behaviour(Crisfield 1979, Matthies and Strang 1979). This lack of numerical stability seems to have decreased the popularity of this class of iterative methods in recent years“.

Somit sind eigene numerische Studien angebracht, um die Robustheit des Verfahrens für die spezielle Klasse von Problemen, die man lösen will, zu überprüfen.

NEWTON/RAPHSON Methoden

λ
V
konstante Lastinkremente: Kraftsteuerung

Grundsätzliche Schwächen der kraftgesteuerten NEWTON/RAPHSON-Methoden:

- Extrema können nicht überwunden werden.
- Die Auflösung der Verformungsachse verschlechtert sich bei aufweichenden Strukturen.

Bogenlängen-Verfahren

Grundidee der Bogenlängenverfahren

linearer Prediktorschritt

λ

Iterationspfad

Gleichgewichtszustände

Iteration erfolgt auf einer Kurve

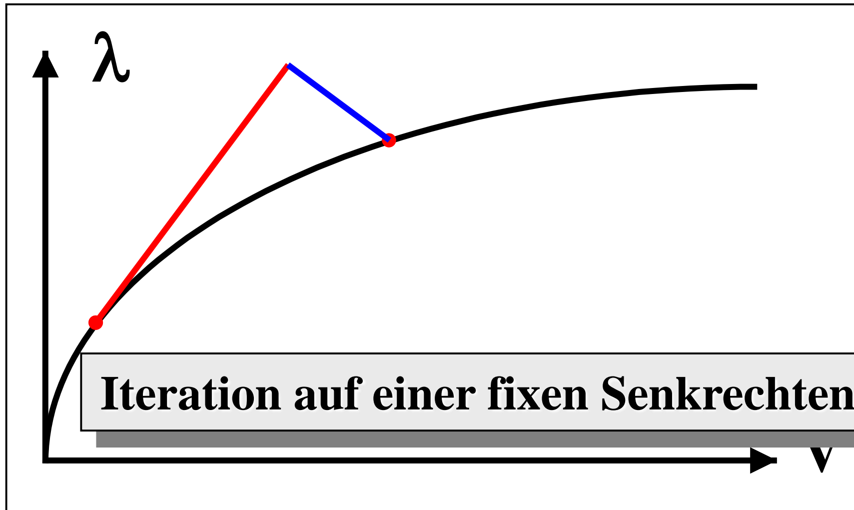
V

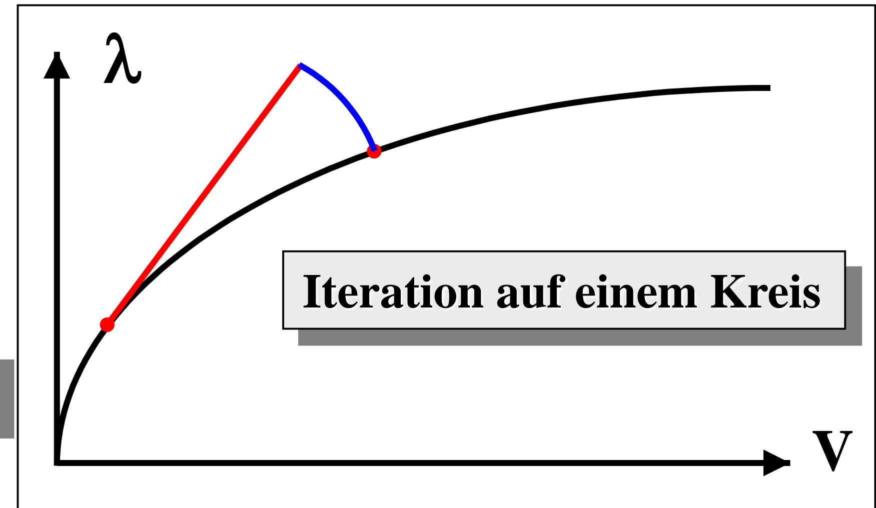
☹ **Bemessungszwecke: Lastniveau kann nicht kontrolliert werden.**

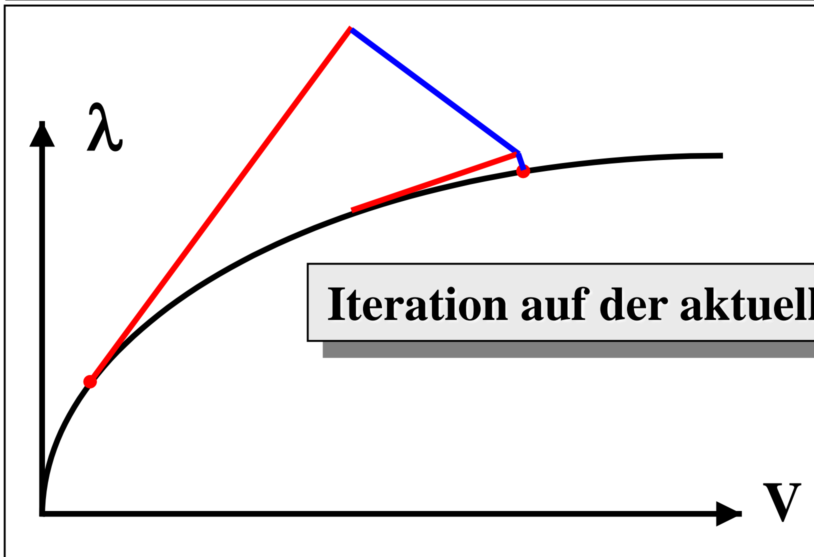
☺ **Pfadverfolgung: Extrempunkte können überwunden werden.**

☺ **Pfadverfolgung: Die Kurve kann mit gleichbleibender Auflösung abgefahren werden.**

Einige mögliche Iterationspfade







menum

RIKS/WESSELS-Methode

λ

$\overset{+}{V}_0$

$\overset{+}{V}_{1u}$

p^*

t^*

p'

0

$\Delta\lambda_1$

n_1

1

m

n_0

$\Delta\lambda^*$

$\Delta\lambda_0$

t_0

p

$\overset{+}{V}_1$

$\overset{+}{V}^*$

Grundzustand

V

Einstieg

Die Iteration auf einer fixen Senkrechten stellt die Urform der Bogenlängenverfahren dar. Sie wurde von RIKS speziell zur Überwindung eines Maximums entwickelt. In der Urformulierung ging die Symmetrie der Steifigkeitsmatrix verloren, wodurch die Standardspeichertechnik und die darauf aufbauenden speziellen symmetrischen Gleichungslöser nicht mehr anwendbar waren. In der Modifizierung von WESSELS konnte durch Gleichungslösung für eine weitere rechte Seite die Symmetrie der Matrix erhalten werden.

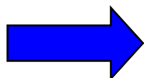
Ausgangspunkt: ausiterierter Grundzustand

Aufbringung eines Lastinkrementes $\Delta\lambda_0$

$\overset{+}{\mathbf{V}}_0$ → **Punkt 0**

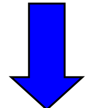
$$\mathbf{t}_0 = \begin{bmatrix} \overset{+}{\mathbf{V}}_0 \\ \Delta\lambda_0 \end{bmatrix}$$

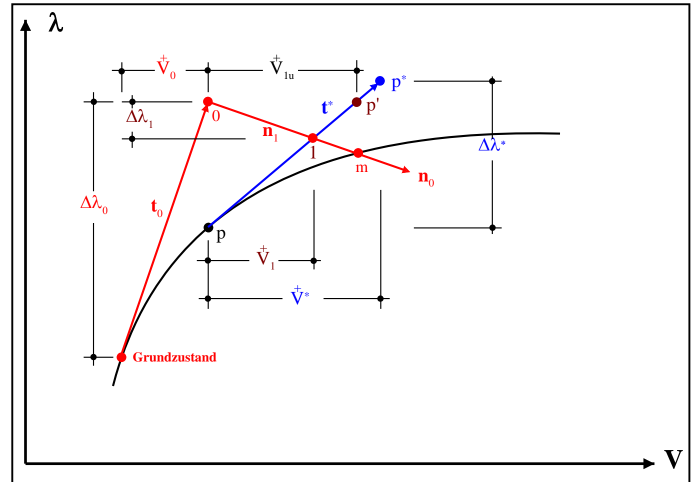
Tangentenvektor t_0

Ungleichgewichtskräfte in p →  → Punkt p'

Mit p' ist ***kein*** Lastfaktor verknüpft!! Die Ungleichgewichtskräfte sind ***nicht*** proportional zu der Referenzlast: $P_u \neq \lambda \cdot P_{ref}$.

Aufbringung eines willkürlichen Lastinkrements Δλ* in p →





$$\mathbf{t}_1 = \begin{bmatrix} \overset{+}{\mathbf{V}}{}^* \\ \Delta\lambda^* \end{bmatrix}$$

Tangentenvektor t_1

$$\mathbf{n}_1 = \begin{bmatrix} \overset{+}{\mathbf{V}}_{1u} \\ 0 \end{bmatrix} + \alpha \begin{bmatrix} \overset{+}{\mathbf{V}}{}^* \\ \Delta\lambda^* \end{bmatrix}$$

Normalenvektor n_1

α: unbekannt

Die Urtangente t_0 und die Normale n_1 stehen senkrecht aufeinander

Skalarprodukt

$\mathbf{t_0 \cdot n_1 = 0}$

$$\begin{bmatrix} \overset{+}{\mathbf{V}}_0 \\ \Delta\lambda_0 \end{bmatrix} \cdot \left\{ \begin{bmatrix} \overset{+}{\mathbf{V}}_{1u} \\ 0 \end{bmatrix} + \alpha \begin{bmatrix} \overset{+}{\mathbf{V}}^* \\ \Delta\lambda^* \end{bmatrix} \right\} = 0$$

Lastkorrektur

$$\alpha = -\frac{\overset{+}{\mathbf{V}}_0 \, \overset{+}{\mathbf{V}}_{1u}}{\overset{+}{\mathbf{V}}_0 \, \overset{+}{\mathbf{V}}^* + \Delta\lambda_0 \, \Delta\lambda^*}$$

Punkt 1 auf der Normalen

$$\overset{+}{\mathbf{V}}_1 = \overset{+}{\mathbf{V}}_{1u} + \alpha \, \overset{+}{\mathbf{V}}^*$$

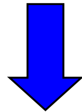
$$\Delta\lambda_1 = \alpha \, \Delta\lambda^*$$

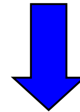
In der Iteration verändert sich das Lastniveau!

Bogenlängenverfahren: kombinierte Kraft-/Wegsteuerung

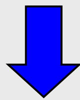
Bogenlänge ΔL

$$\Delta L = \sqrt{\Delta\lambda^2 + \Delta \mathbf{V}^2}$$

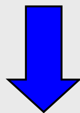




große Referenzlast

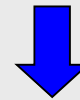


kleiner Lastfaktor

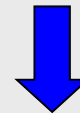


Wegsteuerung

kleine Referenzlast



großer Lastfaktor



Kraftsteuerung

Bogenlängenverfahren: Zusammenfassung

Analog zur Ableitung der Iteration auf einer fixen Senkrechten können Lastkorrektur und verbleibendes Verschiebungsinkrement für Iterationen auf anderen Iterationskurven abgeleitet werden. Die Urform des Bogenlängenverfahrens kann Schwierigkeiten bei großen lokalen Krümmungen bzw. Knicken in der Last-Verformungskurve bekommen. Es können zwar auch dann die Extrempunkte überwunden werden, aber der dann gefundene Gleichgewichtspunkt liegt sehr weit weg vom letzten Punkt, so dass die Kurve in dem dadurch entstehenden Zwischenbereich nicht berechnet wird.

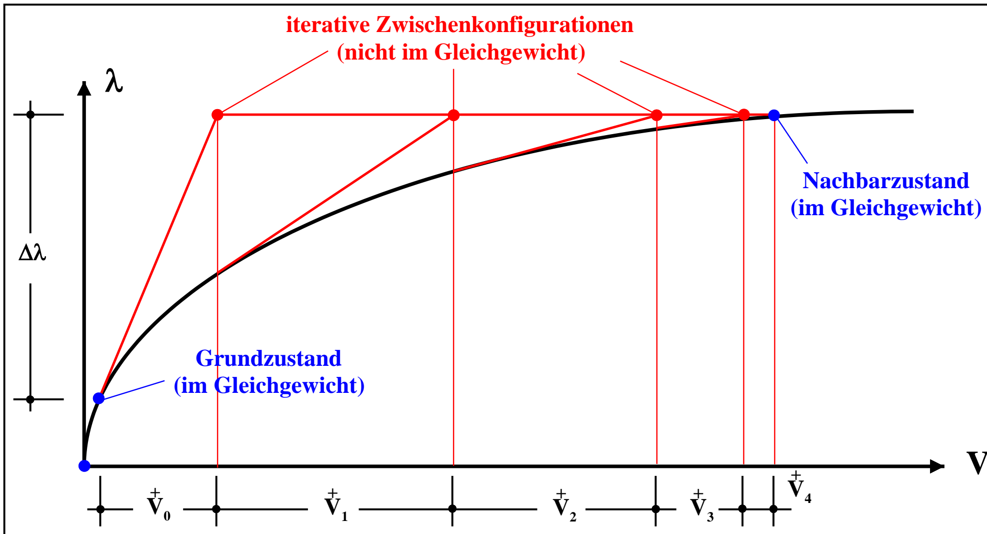
Einen schnelleren Abstieg auf die Kurve erreichen dann die Iterationen auf der aktuellen Normalen oder auf dem Kreisbogen. Auf die Ableitung der zugehörigen Gleichungen wird hier verzichtet – jeder Kursteilnehmer kann diese Ableitung selbst versuchen und danach durch Implementierung die Richtigkeit der Herleitung und die Leistungsfähigkeit überprüfen.

Analog zu den Newton/Raphson-Verfahren können auch die Bogenlängenverfahren in modifizierter Form angewendet werden, indem die Faktorisierte des Laststeigerungsschrittes für alle weiteren Iterationen übernommen wird.

Nach Überwindung eines Extremwertes muss zur weiteren Verfolgung der Kurve die Laststeigerungsrichtung umgedreht werden. Hierfür sind geeignete Steuermechanismen zu implementieren.

Line Search

Grundidee



Im Verlaufe eines Iterationszyklus arbeitet man sich immer näher an das gesuchte Ergebnis heran. Beim Beispiel des obigen aufweichenden Systems sind die Verschiebungsinkremente immer zu klein, da während des linearisierten Iterationsschrittes ein weiterer, durch die Linearisierung nicht erfasster, Steifigkeitsverlust eintritt. Man würde schneller ans Ziel gelangen, wenn man anstatt der errechneten Zuwächse einfach ein „geeignetes“ Vielfaches addieren würde. Das ist die Idee des sog. „line search“, für den es keine gute kompakte deutsche Vokabel gibt.

Frage: wie findet man diesen „geeigneten“ Faktor?

Line Search: Ansatz

Bei einem line search werden die errechneten Inkremente nicht addiert sondern als Richtung aufgefasst, in der die gesuchte Lösung liegen muss. Der „geeignete“ Faktor wird so ermittelt, dass die Energie in dieser Richtung ein Minimum annimmt. Minimale Energie heißt Gleichgewicht. Da sich während des Iterationszyklus das Verhältnis der Verformungskomponenten untereinander ändert, ändert sich auch die Richtung. Man wird also nicht das exakte Gleichgewicht finden, hofft aber, näher an dieses heranzukommen und damit die Iterationszahl senken zu können.

In der Iteration j ermittelter Verschiebungszuwachs

$$\mathbf{K}_j \Delta \mathbf{V}_j = \mathbf{P} - \mathbf{F}_j = \mathbf{P}_{u,j} \quad \Rightarrow \quad \Delta \mathbf{V}_j$$

Wir nehmen an, dass sich das Verhältnis der Verschiebungskomponenten untereinander nicht ändert, und addieren ein um einen Faktor c vergrößertes Verschiebungsinkrement auf:

$$\Delta \tilde{\mathbf{V}}_j = c \cdot \Delta \mathbf{V}_j \quad \Rightarrow \quad \mathbf{V}_{j+1} = \mathbf{V}_j + \Delta \tilde{\mathbf{V}}_j = \mathbf{V}_j + c \cdot \Delta \mathbf{V}_j$$

Bedingung für den Faktor c

Für den „optimalen" Faktor c soll gelten, dass das Gesamtpotential Π im Nachbarzustand V_{j+1} minimal wird.

$$\Pi(\mathbf{V}_{j+1}) = \Pi(\mathbf{V}_j + c \cdot \Delta\mathbf{V}_j) = \min$$

Hierfür wird das Gesamtpotential in einer TAYLOR-Reihe um den Nachbarzustand entwickelt

$$\Pi(\mathbf{V}_{j+1}) = \Pi(\mathbf{V}_j + \Delta\tilde{\mathbf{V}}_j) = \Pi(\mathbf{V}_j) + \frac{\partial \Pi}{\partial \mathbf{V}} \cdot (\Delta\tilde{\mathbf{V}}_j) + \ldots$$

Das lineare Glied in der Reihe entspricht der 1. Variation des Gesamtpotentials. Die Forderung nach dem Verschwinden der 1. Variation ist identisch zur Forderung nach Gleichgewicht. Somit liefert die 1. Ableitung nach dem diskreten Freiheitsgradvektor die diskrete Gleichgewichtsbedingung, die multipliziert mit der Variation der Verschiebung Null ergeben muss.

$$\frac{\partial \Pi}{\partial \mathbf{V}} \cdot (\Delta\tilde{\mathbf{V}}_j) = (\mathbf{P} - \mathbf{F}_{j+1})^T \Delta\tilde{\mathbf{V}}_j = 0$$

Ermittlung des Faktors c

Der Faktor c muss so bestimmt werden, dass das Produkt der Ungleichgewichtskräfte im Pseudo-Gleichgewichtszustand mit dem Verschiebungszuwachs Null ergibt. Ungleichgewichtskräfte sowie Verschiebungszuwachs hängen von dem unbekannten Faktor c ab, wobei die Abhängigkeit der Ungleichgewichtskräfte implizit über die Gesamtverschiebung entsteht. $\mathbf{V}_j$ und $\Delta\mathbf{V}_j$ sind bekannt.

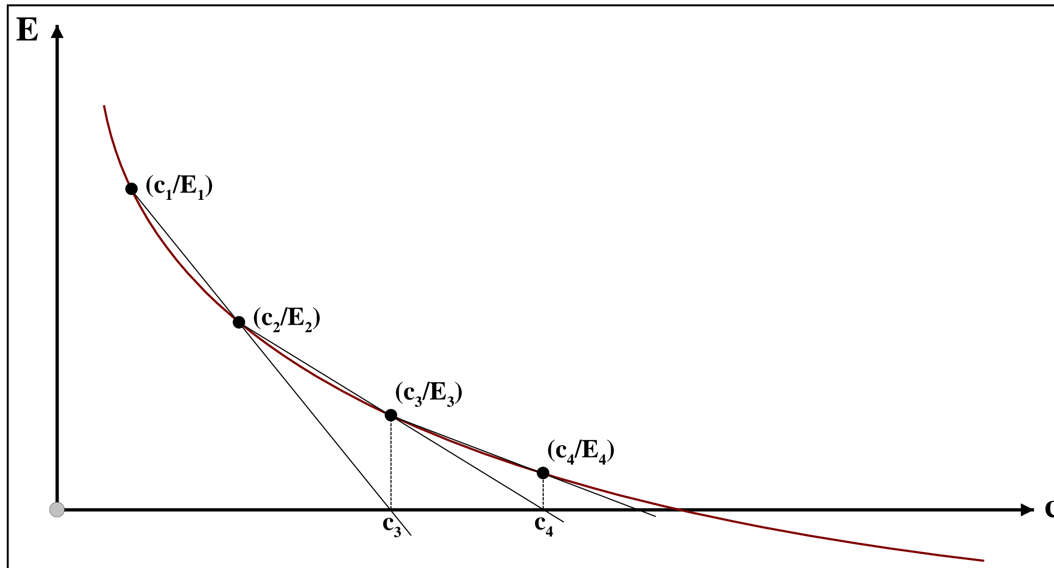
$$E = (\mathbf{P} - \mathbf{F}_{j+1}(\mathbf{V}_{j+1}))^T (\alpha \cdot \Delta\mathbf{V}_j) \approx 0$$

$$\mathbf{V}_{j+1} = \mathbf{V}_j + c \cdot \Delta\mathbf{V}_j$$

Mangels einer expliziten Gleichung für den Faktor c kann dieser nur durch gezieltes Ausprobieren gefunden werden. Hierfür ist es notwendig, in einem Line-Search-Iterationszyklus, der in die eigentliche Iteration eingebettet ist, einen ausreichend guten Faktor zu finden, für den der Energiefehler E ausreichend klein wird.

MENUM

Line Search: Iterationsalgorithmus



Man startet von einem sinnvollen Ausgangswert für c, beispielsweise 1, und errechnet hierfür den Energiefehler E. Dann wird c verändert und erneut E berechnet. Ist das neue E ausreichend klein im Vergleich zu dem alten, ist die Iteration zu Ende. Wenn nicht, kann ein neuer Schätzwert für c als Schnittpunkt der Geraden durch die beiden letzten Punkte mit der Abszisse gefunden werden, für den erneut E berechnet wird. Dieser Prozess wird so lange wiederholt, bis E ausreichend klein ist. Da die Berechnung der inneren Kräfte F alle Elemente abarbeitet, welches messbare Rechenzeit verbrauchen kann, sollte die Toleranzgrenze nicht zu klein gesetzt werden, da ansonsten der Zeitgewinn durch weniger Iterationen durch exzessiv viele Line-Search-Iterationen wieder aufgefressen wird.

MENUM

Zusammenfassung & nächste Schritte

In Teil A des Vorlesungsblocks 4 wurden die gängigen Pfadverfolgungsalgorithmen ohne mathematische Strenge abgeleitet. Aus der Ableitung ergaben sich theoretische Vor- und Nachteile. Wie sich die Algorithmen im Praxiseinsatz bewähren, welche faktischen Stärken und Schwächen sie haben, lässt sich nur durch umfangreiche Testrechnungen herausarbeiten. Hierbei hängt die Leistungsfähigkeit immer auch von der Implementierung ab, so dass unterschiedliche Programme sich auch unterschiedlich verhalten können.

Somit ergibt sich der nächste Schritt praktisch von selbst: Wir werden an unterschiedlichen Beispielen die Algorithmen einem Testprogramm unterziehen.