Ecole Nationale des Ponts et Chaussées

2004

Wissenschaftliches Praktikum

Jérémie Simon

Aurélien Boffy

Eine Hilfe zum Verständnis der Finite-Elemente-Methode FEJavaDemo

Vom 5. April bis 25. Juni

Betreuer: Doz. Dr. Michael Jung

Praktikumsverantwortliche:

Prof. Dr. Wolfgang Walter (TU Dresden)

H. Renaud Keriven (ENPC)

Danksagungen

Wir beide möchten unserem Betreuer, Doz. Dr. Michael Jung, für seine Verfügbarkeit, seine Hilfe, und insbesondere seine Sympathie während des Praktikums danken. Er beriet uns, indem er unsere Initiativen vertrat.

Wir danken auch Prof. Dr. Wolfgang Walter, Direktor des Instituts für Wissenschaftliches Rechnen, der alles tat, damit unseres Praktikum in Dresden bestens abläuft.

Kshitij Kulshreshtha, Mitarbeiter, half uns, was die Programmierung betrifft, und wir sind sehr dankbar dafür.

Wir bedanken uns auch bei allen Personen des Instituts für Wissenschaftliches Rechnen, insbesondere bei Frau Gudrun Heinisch für ihre Freundlichkeit.

In Frankreich danken wir auch den Personen, die das Praktikum erlaubt oder unterstützt haben:

Herrn Jean-Jacques Colleu und den gesamten Département de la Formation Alternée der ENPC.
Herrn Renaud Keriven, unserem Praktikumsverantwortlichen.
Der Fondation de l'Ecole Nationale des Ponts et Chaussées.

Zusammenfassung

Das Thema des Praktikums ist die Programmierung einer Software (FEJavaDemo) in Java, die Wärmeleitprobleme mittels der Methode der finiten Elemente löst. Das Programm dient zur Illustration des Buches „*Methode der finiten Elemente für Ingenieure*“, von dem unser Betreuer H. Jung der Mitverfasser ist. Es ist ein Lehrprogramm: der Benutzer kann die verschiedenen Prozesse der Assemblierung und der Auflösung dank Zeichentricks und Anzeige der implementierten Algorithmen verfolgen. Lineare oder quadratische Ansatzfunktionen über Dreiecken oder Vierecken werden genutzt, aber FEJavaDemo wurde entwickelt, um später weitere Objektarten zu behandeln. Damit das Programm von vielen Leuten benutzt werden kann, wird es in drei Sprachen übersetzt und mit einem vollständigen Handbuch verknüpft.

Schlüsselwörter: FEM, Methode der finiten Elemente, Wärmeleitproblem, Java, Lehrprogramm, Cuthill McKee, wissenschaftliches Rechnen,

Inhaltsverzeichnis

1. Präsentation unseres Praktikum

1.1. Geschichte der Universität

Die Technische Universität Dresden gehört zu den ältesten Technischen Universitäten Deutschlands.

Als die Technische Bildungsanstalt Dresden - Vorläufer der heutigen Universität im Jahre 1828 in der Elbestadt gegründet wurde, befand sich die industrielle Revolution in Sachsen erst in den Anfängen.
Doch mit dieser neuen Bildungsstätte erhielt Sachsen die Möglichkeit, dringend benötigte Mechaniker und Techniker für die einheimische Industrie auszubilden und so die industrielle Entwicklung zu fördern.

1871 erhielt sie die Anerkennung als Königlich-Sächsisches Polytechnikum, ein wichtiger Schritt auf dem Weg zur Technischen Hochschule.
Schon damals war eine Allgemeine Abteilung eingerichtet, in der Lehrstühle für Nationalökonomie und Statistik, Deutsche Sprache und Literatur sowie Kunstgeschichte vereinigt waren.

1865 begann die Ausbildung von Lehrern für Mathematik, Naturwissenschaft und Technik. Die Fachabteilungen Maschinenbau, Bauwesen und Chemie hatten sich bald wissenschaftlich profiliert und verhalfen der Schule über die Grenzen Sachsens hinaus zu Ansehen.

Dem wurde schließlich 1890 mit der Verleihung des Status Technische Hochschule und 1900 mit den Promotionsrechten auch amtlich entsprochen. Das Ende des 19. und die ersten Jahrzehnte des 20. Jahrhunderts brachten der Hochschule ihre entscheidende wissenschaftliche Profilierung. Neue Institute waren mit dem Wirken vieler bedeutender Gelehrter an der Hochschule verbunden.

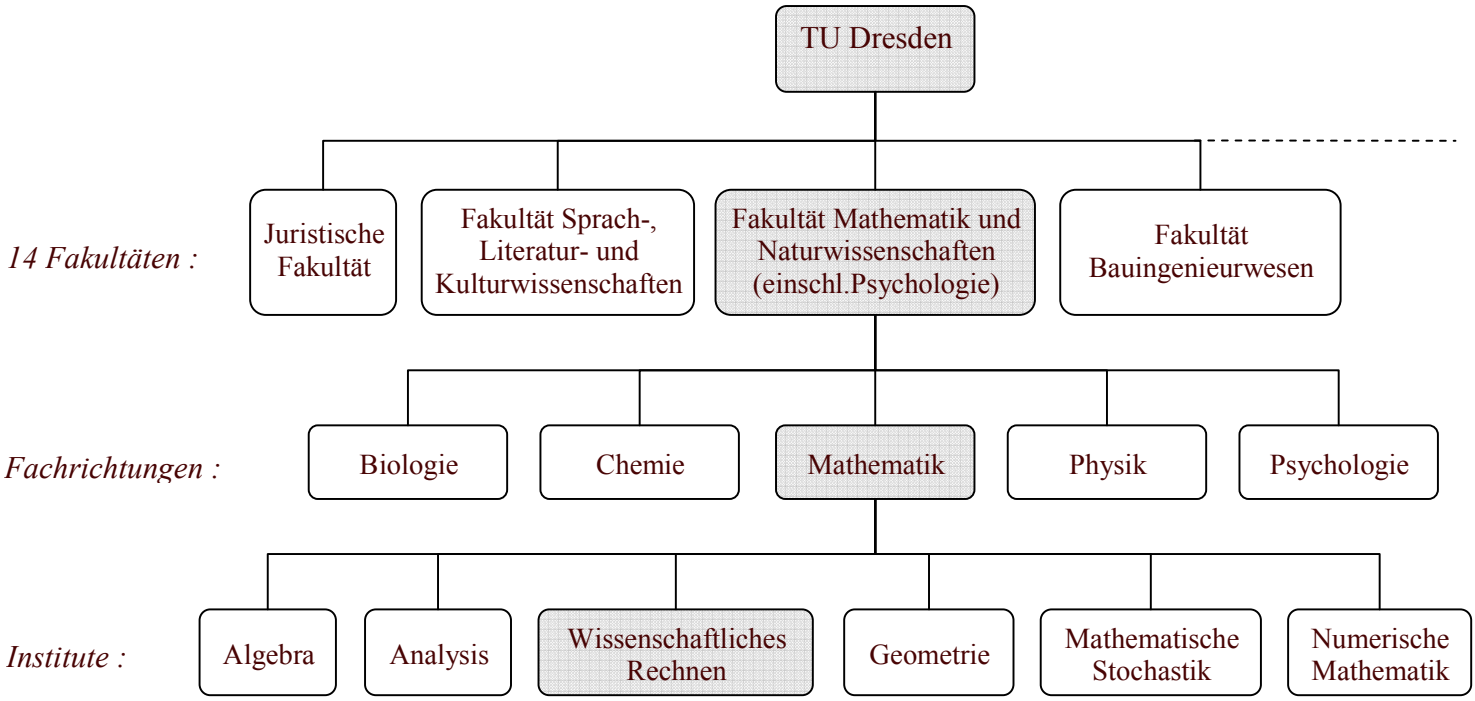
Das wissenschaftliche Profil der Technischen Hochschule Dresden hatte immer mehr universellen und damit universitären Charakter. Neben der großen Breite ingenieurwissenschaftlicher Disziplinen gehörten auch Wirtschafts- und Geisteswissenschaften, die Pädagogik und natürlich die Naturwissenschaften zum wissenschaftlichen Potential dieser Hochschule. Dem ist 1961 durch die neue Bezeichnung Technische Universität Dresden Rechnung getragen worden.

Seit der deutschen Wiedervereinigung wurden den traditionellen natur- und ingenieurwissenschaftlichen Fachgebieten neue Fakultäten im Bereich der Geistes-, Sozial- und Wirtschaftswissenschaften sowie der Medizin hinzugefügt. Damit verfügt die Technische Universität Dresden heute über ein wissenschaftliches Spektrum in Forschung und Lehre, das in seiner Breite und Vielfalt nur an wenigen Universitäten Deutschlands zu finden ist.

Mit 30500 Studierenden und rund 4 500 Mitarbeitern (ohne Medizinische Fakultät) - darunter rund 480 Professoren - ist sie heute die größte Universität Sachsens (Stand 1.12.2002). Seit 1994 ist die TU Dresden eine Volluniversität mit 14 Fakultäten. Damit umfasst das wissenschaftliche Profil der Universität alle Bereiche der Ingenieurwissenschaften, Geistes- und Sozialwissenschaften, Naturwissenschaften und der Medizin. Neben dem breiten Spektrum der Studienangebote sind 16 internationale Studiengänge eingerichtet worden.
Die interdisziplinäre Zusammenarbeit aller Fachrichtungen gehört zu den Stärken der TU Dresden. In Lehre und Forschung gilt das Prinzip, Studierende und Diplomanden frühzeitig in aktuelle Forschungsaufgaben einzubeziehen. Die Beziehungen zur Wirtschaft sind vielfältig. Die TU Dresden fördert besonders Erfinder an der Universität, sichert ihre Schutzrechte und sorgt für den schnellen Transfer der Erfindungen in marktfähige Produkte.
Für ihre Absolventen und alle Interessierten gestaltet die TU Dresden praxisgerechte, individuell konzipierte Weiterbildung. Über 20 Aufbaustudiengänge, auch mit internationalem Abschluss, blended learning und Karriereberatung gehören zum Weiterbildungsangebot der Universität.

1.2. Das Institut für Wissenschaftliches Rechnen

Wir haben am Institut für Wissenschaftliches Rechnen gearbeitet, das zur Fachrichtung Mathematik gehört. Diese Sektion Mathematik wurde im Jahr 1968 gegründet und besteht heute aus sechs Instituten.



Einige Schwerpunkte der Forschung des Instituts sind zum Beispiel:

- Identifikation und Optimierung auf der Basis von Sensitivitätsberechnungen
- Ergebnisverifikation, Computerarithmetik und wissenschaftliche Programmiersprachen
- Multilevel-Verfahren bei der Simulation physikalisch-technischer Prozesse
- Paralleles Rechnen und Software-Werkzeuge
- Computergraphik und Visualisierung

Der Betreuer unseres Praktikums heißt Michael Jung. Er ist Hochschuldozent für Wissenschaftliches Rechnen. Seine Forschungsgebiete sind unter anderem:

- Konvergenztheorie von Multilevel-Verfahren
- Konstruktion von Vorkonditionierungsoperatoren mittels Multilevel-Techniken
- Kombination von Adaptivität und Multilevel-Verfahren
- Implementierung von Multilevel-Verfahren auf Parallelrechnern
- Anwendung von Multilevel-Verfahren auf Probleme aus der Praxis
- Kopplung von Finite-Elemente- und Randelemente-Methoden
- Mehrfeldprobleme

Sein Fachgebiet ist "Multilevel-Verfahren und Parallelisierung" und er gibt Vorlesungen, insbesondere, über die Finite-Elemente-Methode für Ingenieurstudenten und Numerik partieller Differentialgleichungen.

1.3. Unser Praktikumsthema

1.3.1. Das Buch unseres Betreuers

Michael Jung hat mit seinem Kollegen Ulrich Langer, der an der Johannes Kepler Universität Linz (Österreich) arbeitet, ein Buch über finite Elemente geschrieben: *Methode der finiten Elemente für Ingenieure. Eine Einführung in die numerischen Grundlagen und Computersimulation.* Dieses Lehrbuch entstand auf der Grundlage von Vorlesungen zum „Thema Numerik partieller Differentialgleichungen", welche die Autoren in den letzten Jahren für Ingenieurstudenten gehalten haben. Das Lehrbuch ist als eine Einführung in die numerische Lösung partieller Differentialgleichungen und in das dazu notwendige Handwerkszeug aus der Numerischen Mathematik, das an den entsprechenden Stellen eingespielt wird, gedacht. Der Weg von der physikalischen Erscheinung zum mathematischen Modell wird am Beispiel der stationären Wärmeleitung beschrieben. Die hergeleiteten Differentialgleichungen dienen als Modellbeispiele bei der Beschreibung der FEM.
Wir haben uns besonders für das 4. Kapitel interessiert, das der FEM für elliptische Randwertaufgaben in mehrdimensionalen Gebieten gewidmet ist. Zunächst beschreibt es allgemein das Ritz- und Galerkin- Verfahren als mögliche Diskretisierungsverfahren.

Anschließend wird die Methode der finiten Elemente als spezielles Ritz-Galerkin-Verfahren erläutert. Es werden die Teilschritte der Finite-Elemente-Diskretisierung im Detail beschrieben und ausführlich am Beispiel der linearen Dreieckselemente illustriert.

Auf der Homepage zum Lehrbuch findet der an der Implementierung der FEM interessierte Leser das Finite-Elemente-Programm FEM2D zum Herunterladen sowie eine Sammlung von Übungs- und Praktikumsaufgaben zur Vorlesung. Das Programm veranschaulicht das Buch, indem es die gleichen Beispiele benutzt und ermöglicht, die verschiedenen Teilschritte der FEM zu demonstrieren. Mit dem Programm kann man Wärmeleitprobleme lösen.
Dieses Lehrprogramm wird von Studenten genutzt, und es kann auch dem Lehrenden helfen, einen Kurs über FEM vorzubereiten, indem es Jpeg Bilder der Vernetzung und der Matrix erstellen kann.
Es bleibt aber ein Lehrprogramm und kann natürlich nicht mit einem „richtigen“ FEM-Programm konkurrieren, was die Geschwindigkeit der Rechnungen betrifft. Dennoch integriert es Methoden, um die Lösung zu beschleunigen. Unter anderem erfasst es den Fakt, dass die Matrix schwach besetzt ist.

Das Ziel unseres Praktikums war diese Software neuzuprogrammieren, weil sie Nachteile hat:

- Sie ist nicht modular, d.h. man kann keinen Bestandteil hinzufügen. Zum Beispiel kann das Programm nur mit linearen Ansatzfunktionen über Dreiecken arbeiten.
- Sie kann nur unter Windows laufen.
- Das Interface ist stark verbesserbar.
- Die Programmiersprache (Fortran) wird seltener genutzt als Java oder C++.

1.3.2. Das Wärmeleitproblem

Gesucht ist das stationäre Temperaturfeld $u(x)$ in einem zweidimensionalen Gebiet Ω, das aus mehreren Materialbereichen mit verschiedenen Wärmeleitzahlen $\Lambda(x)$ bestehen kann. Im Gebiet kann es eine Wärmequelle geben, die durch eine Funktion f beschrieben wird.

Drei Arten von Randbedingungen können betrachtet werden:

- Die Dirichletschen Randbedingungen (Randstück Γ_1): Teil des Gebietsrandes, auf dem Werte $g_1(x)$ für die Temperatur vorgegeben sind.
- Die Neumannschen Randbedingungen (Randstück Γ_2): Teil des Gebietsrandes, auf dem der Wärmefluß $g_2(x)$ vorgegeben ist (Mit $g_2(x)=0$ kann auf Randstücken eine Wärmeisolation modelliert werden)
- Die Robinschen Randbedingungen (Randstück Γ_3): Modellierung des freien Wärmeaustauschs mit der Umgebung (der Wärmefluß ist proportional ($\alpha(x)$ Koeffizient) zur Differenz zwischen dem Wert der Temperatur am Rand $u(x)$ und der Umgebungstemperatur $u_A(x)$)

Klassische Formulierung:

Gesucht ist $u \in C^2(\Omega) \cap C^1(\Omega \cup \Gamma_2 \cup \Gamma_3) \cap C(\bar{\Omega})$, so dass

$$
\begin{aligned}
\text{-div}(\Lambda(x)\ \text{grad}\ u(x)) &= f(x) && \forall x \in \Omega\ , \\
u(x) &= g_1(x) && \forall x \in \Gamma_1\ , \\
\frac{\partial u}{\partial N} &= g_2(x) && \forall x \in \Gamma_2\ , \\
\frac{\partial u}{\partial N} &= \alpha(x)\ \left[u_A(x) - u(x)\right] && \forall x \in \Gamma_3\ ,
\end{aligned}
$$

gilt mit $\Gamma = \partial\Omega = \bar{\Gamma}_1 \cup \bar{\Gamma}_2 \cup \bar{\Gamma}_3$ und $\Gamma_i \cap \Gamma_j = \emptyset$ für $i \neq j$

und $\Lambda(x) = \begin{pmatrix} \lambda_1(x) & 0 \\ 0 & \lambda_2(x) \end{pmatrix}$ mit $\frac{\partial u}{\partial N} = \lambda_1(x)\frac{\partial u}{\partial x_1} n_1 + \lambda_2(x)\frac{\partial u}{\partial x_2} n_2$

und $\vec{n}(x) = \begin{pmatrix} n_1(x) \\ n_2(x) \end{pmatrix}$ der Vektor der äußeren Einheitsnormalen im Punkt x $\in \Gamma$.

Variationsformulierung der Aufgabe:

Gesucht ist $u \in V_{g_1}$, so dass

$$a(u,v) = \langle F, v \rangle \qquad \forall v \in V_0$$

gilt mit

$$a(u,v) = \int_\Omega (\text{grad } v(x))^T \Lambda(x) \text{ grad } v(x)\, dx + \int_{\Gamma_3} \alpha(x) u(x) v(x) ds ,$$

$$\langle F, v \rangle = \int_\Omega f(x) v(x) dx + \int_{\Gamma_2} g_2(x) v(x) ds + \int_{\Gamma_3} \alpha(x) u_A(x) v(x) ds ,$$

$$V_{g_1} = \left\{ u \in H^1(\Omega) : u(x) = g_1(x) \text{ auf } \Gamma_1 \right\},$$

$$V_0 = \left\{ v \in H^1(\Omega) : v(x) = 0 \text{ auf } \Gamma_1 \right\}.$$

1.3.3. Die geplante Arbeit

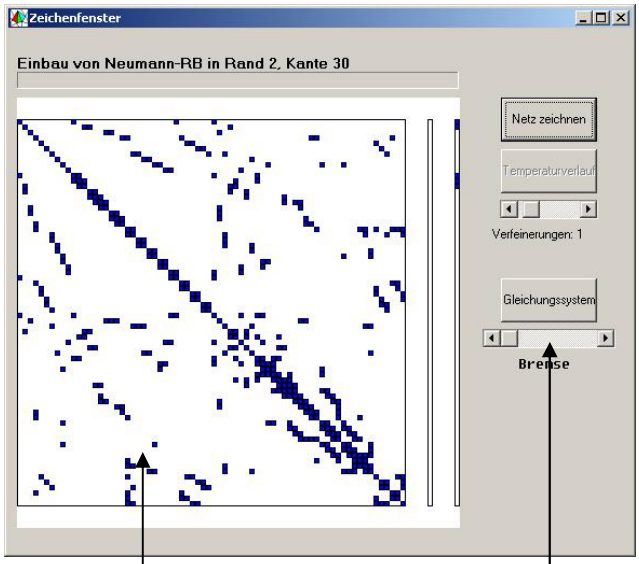
Das Programm, das wir geplant haben, sollte solche Wärmeleitprobleme behandeln.
Mindestens sollte die Software mit linearen Ansatzfunktionen über Dreiecken arbeiten, und sollte anpassungsfähig sein, um später andere Arten von Funktionen und Elementen zu integrieren.

Um ein Problem zu lösen, brauchte das ehemalige Programm zwei Dateien: [siehe Anlage 2: Struktur der Angabefiles]

- ein Netz-File, das die Informationen über die Vernetzung enthält (d.h. die Knotenkoordinaten, die Eckpunkte jedes Elementes, usw.)
- ein Daten-File, das die Wärmeleitzahlen der Materialbereiche, die Art der Randbedingungen in den Randbereichen und die zugehörigen Randdaten, und die Funktion f enthält.

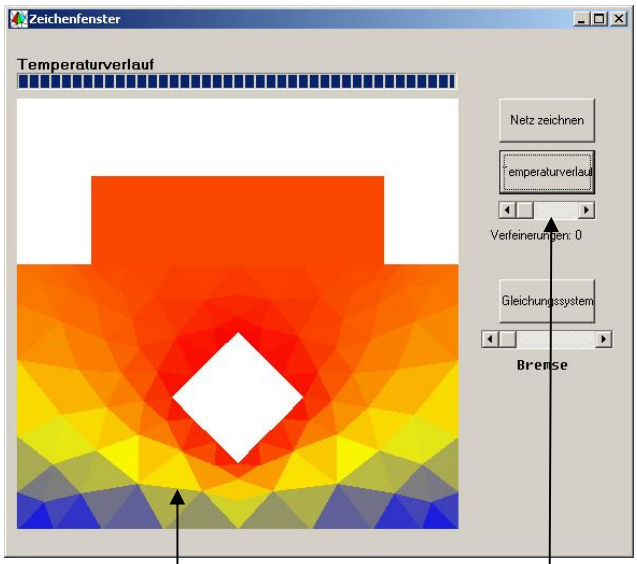
Das neue Programm soll kompatibel mit diesen Dateien sein, um auch mit den ehemaligen Beispielen arbeiten zu können.

Bildschirmfotos des ehemaligen Programms:



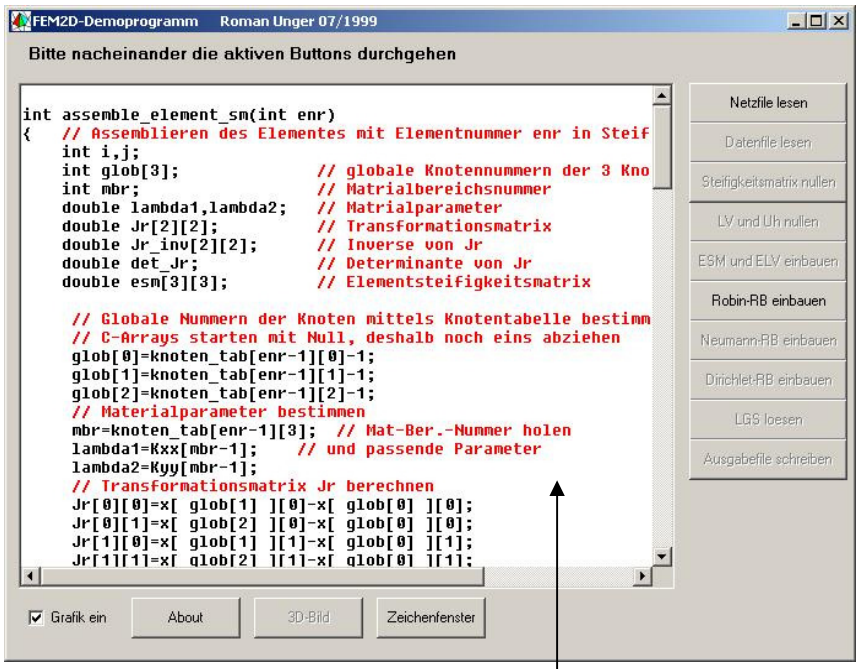
Matrix und Lastvektor Ansehen

Scroll bar, um die Anzeige zu verlangsamen



Lösung

Scroll bar, um die Vernetzung zu verfeinern (nur die Farben)



Knöpfe, um die verschiedenen Schritte der Problembearbeitung durchzuführen

Algorithmen

Wie das ehemalige Programm soll unsere Software die verschiedenen Teilschritte der FEM demonstrieren. In einem Fenster werden die Steifigkeitsmatrix und der Lastvektor mit eingefärbten Nicht-Null-Einträgen dargestellt, um zu zeigen, dass die Matrix schwach besetzt ist. Dann kann man die Assemblierung schrittweise durchführen, d.h. zuerst die Assemblierung ohne Randbedingungen und dann die Berücksichtigung der Randbedingungen.

In einem anderen Fenster kann man den Algorithmus lesen, der für den jeweiligen Schritt benutzt wird. Die Vernetzung und die Lösung werden ebenfalls dargestellt. Um die Lösungsdarstellung zu verbessern, können intern die Dreiecke wiederholt in vier Teildreiecke zerlegt werden und die Lösung wird über ihnen linear interpoliert.

Herr Jung wünschte auch ein Handbuch für die Benutzer und eine Dokumentation des Programms, falls jemand anderes Funktionalitäten hinzufügen will.
Eine Möglichkeit, Jpeg Bilder der Vernetzung und der Matrix zu erstellen, war auch beabsichtigt.

Wir haben mit Herrn Jung entschieden, Java zu benutzen, um die vorher genannten Probleme zu lösen. Diese Programmiersprache ist nämlich plattformunabhängig, frei verfügbar, leicht zu lernen (nicht viele Unterschiede mit C++), sehr internet-freundlich (in www-Seiten eingebettete Applets. Ein Applet ist ein dynamisches interaktives Programm, das aus dem World Wide Web in einen Web-Browser heruntergeladen werden kann und in einer HTML-Web-Page abläuft) und mit eingebauter GUI (Graphical User Interface): alles was man zur Gestaltung graphischer Benutzenoberfläsche braucht. Dazu ist Java echt objektorientiert und in der Zukunft wird man einfach neue Funktionalitäten hinzufügen, ohne den ganze Code zu ändern.

2. Durchführung der Arbeit

2.1. Einige Wiederholungen zur FEM

Die Methode der finiten Elemente ist ein numerisches Verfahren zur näherungsweisen Lösung von partiellen Differentialgleichungen mit Randbedingungen.

Bei der FEM wird das Gebiet Ω, in dem die Lösung des betrachteten Randwertproblems gesucht wird, in „kleine“, nicht überlappende Elemente $T^{(r)}$ zerlegt. Als Ansatzfunktionen wählt man für Randwertprobleme 2. Ordnung stetige, stückweise polynomiale Funktionen, die nur über sehr wenigen dieser Teilgebiete von Null verschieden sind. Durch eine Linearkombination der n Ansatzfunktionen werden die möglichen Lösungen der numerischen Näherung festgelegt.
Bei FE-Diskretisierungen für Randwertprobleme in zweidimensionalen Gebieten werden im Allgemeinen Dreiecks- oder Viereckselemente genutzt. Dabei wählt man meist Ansatzfunktionen, die über den Dreiecken lineare, quadratische oder kubische Funktionen sind.

Ausgangspunkt: Variationsformulierung (siehe (1.1))

Gesucht ist $u \in V_{g_1} \;:\; a(u,v) = \langle F, v \rangle \qquad \forall v \in V_0.$ (2.1)

Die Grundidee des Galerkin-Verfahrens besteht darin, den unendlichdimensionalen Raum der Grundfunktionen durch einen endlichdimensionalen Raum V_h zu ersetzen:

$$V_h = \left\{ v_h \;:\; v_h = \sum_{i \in \bar{\omega}_h} v_i p_i(x) \right\} = span\{ p_i : i \in \bar{\omega}_h \} \quad \subset \quad V = H^1(\Omega)$$

Dabei bezeichnet $\bar{\omega}_h$ die Indexmenge, welche die Nummern aller Knoten enthält.

Die vorgegebenen Funktionen p_i sind die so genannten linear unabhängigen Ansatzfunktionen und die Koeffizienten v_i sind frei wählbar.

Da wir die Werte der Lösung in den Dirichlet-Knoten kennen, suchen wir eine Näherungslösung der Aufgabe in der Form

$$u_h(x) = \sum_{j \in \omega_h} u_j p_j(x) \; + \sum_{j \in \gamma_h} u_{*,j} p_j(x)$$

wo nur die u_j unbekannt sind.

Hierbei ist ω_h die Indexmenge, welche die Nummern der Knoten in $\Omega \cup \Gamma_2 \cup \Gamma_3$ enthält und γ_h die Indexmenge, welche die Nummern der Knoten auf $\bar{\Gamma}_1$ enthält: $\bar{\omega}_h = \omega_h \cup \gamma_h$.

Damit ergibt sich das Gleichungssystem:

Gesucht ist $\underline{u}_h = [u_j]_{j \in \omega_h} \in \mathbb{R}^{N_h}$:

$$\sum_{j \in \omega_h} u_j a(p_j, p_i) = \langle F, p_i \rangle - \sum_{j \in \gamma_h} u_{*,j} a(p_j, p_i) \qquad \forall i \in \omega_h$$

d.h. gesucht ist $\underline{u}_h = [u_j]_{j \in \omega_h} \in \mathbb{R}^{N_h}$:

$$K_h \underline{u}_h = \underline{f}_h$$

mit

$$K_h = [a(p_j, p_i)]_{i, j \in \omega_h}$$

$$\underline{f}_h = \left[\langle F, p_i \rangle - \sum_{j \in \gamma_h} u_{*,j} a(p_{j,} p_i) \right]_{i \in \omega_h} \in \mathbb{R}^{N_h}$$

N_h ist die Anzahl der Knoten, die in $\Omega \cup \Gamma_2 \cup \Gamma_3$ liegen.

Im Programm wird folgendes für die Eingangsdaten vorausgesetzt:

- Die Funktion f kann auf verschiedenen Bereichen verschiedene Definitionen haben. Dazu kann sie mittels Polynomen, tan(), sin(), cos(), exp(), ln(), sqrt() und der Konstante π definiert werden, zum Beispiel f(x,y) = 2*cos(Pi*x*y).

- Λ wird mittels λ_1 und λ_2 definiert. Diese Koeffizienten sind Konstanten, die verschieden in jedem Materialbereich sein können.

- Was die Dirichletschen Randbedingungen betrifft, wird die Temperatur $g_1(x)$ nur auf einigen Punkten vorgegeben. Diese Punkte sind die so genannten Dirichlet-Knoten.

- Die anderen Funktionen $g_2(x)$, $\alpha(x)$ und $u_A(x)$ in den Randbedingungen 2. und 3. Art sind auf jeder Kante konstant. So kann man auch sagen, dass sie stückweise konstant sind.

2.2. Unser Lernen von Java

Zu Beginn des Praktikums hatten wir beide eine ziemlich gute Erfahrung in der Programmierung und ins Besonderes hatten wir ein Semester einen C++ Kursen der ENPC belegt. Trotzdem kannten wir nicht die Programmiersprache Java. Deshalb haben wir von Anfang an Bücher ([2], [3]) über diese Sprache gelesen.
In kurzer Zeit ist uns klar geworden, dass es nicht viele Unterschiede zwischen Java und C++ gibt. Die beiden sind beispielsweise objektorientiert und die Syntax ist oft vergleichbar.
Dennoch haben wir mit mehr Aufmerksamkeit einige Kapitel gelesen:

- Während der C++ Kurse hatten wir nie den Begriff der Vererbung gesehen. Vererbung bedeutet, dass Objekte Eigenschaften und Methoden von anderen Objekten erben, d.h. übernehmen können. Dies war sehr wichtig für unser Programm. Es war nämlich nötig Basisklassen oder Oberklassen und abgeleiteten Klassen oder Unterklassen zu benutzen, damit das Programm modulierbar ist. Zum Beispiel haben wir einen Basisklasse „Elem“, die ein Element darstellt und zwei abgeleiteten Klassen hat: „Triangle“ (Dreieck) und „Quadrilateral“ (Viereck).

- Auf die gleiche Weise kannten wir nichts über „Multithreading“. Für ein Programm mit graphischer Oberfläche ist es sinnvoll, die Ausführung in mehrere einzelne Prozesse bzw. Threads zu unterteilen, die unabhängig voneinander ablaufen. Damit kann sich ein Thread um die Aktualisierung der Bildschirmausgabe kümmern, während ein anderer Prozess Datenstrukturen manipuliert. Einige Schritte der Auflösung können nämlich lange dauern und es ist notwendig, während dieser Vorgänge mit dem Programm zu interagieren. Zum Beispiel kann man die Assemblierung beschleunigen und verlangsamen, oder einen Verfeinerungsprozess, der zu lange dauert, unterbrechen.

- Wir haben auch entschieden, das Programm in drei Sprachen zu entwickeln. Wir haben einige Methoden probiert, um die Software zu internationalisieren, und wir haben eine gewählt, mit der es einfach ist, neue Sprachen zu hinzufügen. Es genügt, eine Text-Datei zu übersetzen, wo alle Namen der Knöpfe, Menüs, Hilfen, usw. geschrieben sind.

- Das Programm brauchte auch eine richtige graphische Benutzungsoberfläche. Deshalb haben wir viel über „Swing“ gelesen, Das Swing-Paket dient dazu, unter Java grafische Benutzeroberflächen zu erstellen. Dazu bietet es Komponenten wie Fenster, Dialogboxen,

Buttons, Textfelder usw., aus denen eine Oberfläche zusammengestellt werden kann. Des Weiteren gibt es auch noch Möglichkeiten, auf die durch den Benutzer (z.B. durch das Anklicken eines Buttons) ausgelösten Ereignisse zu reagieren.

- Damit das Programm immer mehr kompatibel wird, d.h. das Programm sollte schön, benutzerfreundlich und benutzbar auf allen Bildschirme mit allen Auflösungen laufen. Es gibt grafische Oberflächen, bei denen die Anordnung der Komponenten grundsätzlich über die Angabe absoluter Koordinaten in Pixel erfolgt. Diese Vorgehensweise ist aber aus mehreren Gründen unvorteilhaft. Einer des gravierendsten Nachteils ist, dass das Erscheinungsbild eines Dialogs mit absoluten Koordinaten auf verschiedenen Plattformen stark variieren kann. Durch die Vermeidung absoluter Größenangaben ist es auch möglich, dass eine Komponente ihre Größe beim Verändern der Container-Größe anpassen kann. Deshalb haben wir einen „Layout Manager“ (GridBagLayout) benutzt, der automatisch festlegt, wie Komponenten in einem Fenster angeordnet werden.

- Damit die Software robust wird, sollten wir uns für die Fehlerbehandlung interessieren. Das Programm soll unter fast allen Umständen sicher weiterlaufen und nicht abstürzen. Dieses Konzept wird in Java durch die so genannten Exceptions (Ausnahmen) realisiert. Was unser Programm betrifft, kann man Exceptions haben, wenn man zum Beispiel einen unrichtigen Dateinamen schreibt, um die Vernetzung zu laden. Manchmal kann es auch einen Speicherfehler geben, wenn wir mit großen Vernetzungen arbeiten, die zu verfeinern sind.

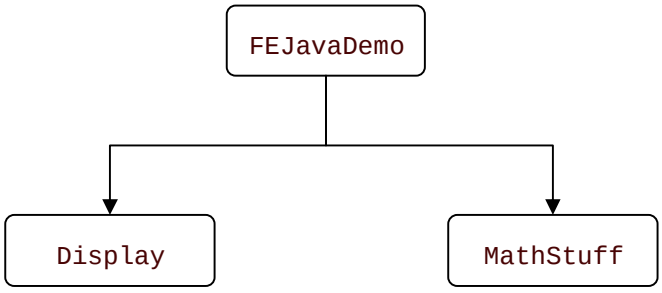
- Herr Jung wünschte auch, dass man im Internet das Programm probieren kann. Deshalb haben wir uns über die „Applets“ erkundigt. Applets sind Java-Programme, die in HTML-Seiten eingebunden werden können. Es ist nicht mehr notwendig, die Software auf dem lokalen Rechner zu installieren. Die Anwendung wird innerhalb des Browsers oder Appletviewer unabhängig vom System des Benutzers ausgeführt. Die Hauptfunktionalitäten des Programms sind auch mit dem Applet benutzbar.

- Das Programm benutzt mehrmals Dateien: er liest Vernetzungen und soll fähig sein, neue .net und .dat Dateien zu schreiben, die Lösung des Problems in einer Text-Datei, zu speichern, und Jpeg-Bilder zu erstellen. Deshalb haben wir auch gelernt, die Informationsflüsse in Java zu benutzen.

2.3. Beschreibung und Gestaltung der Klassen

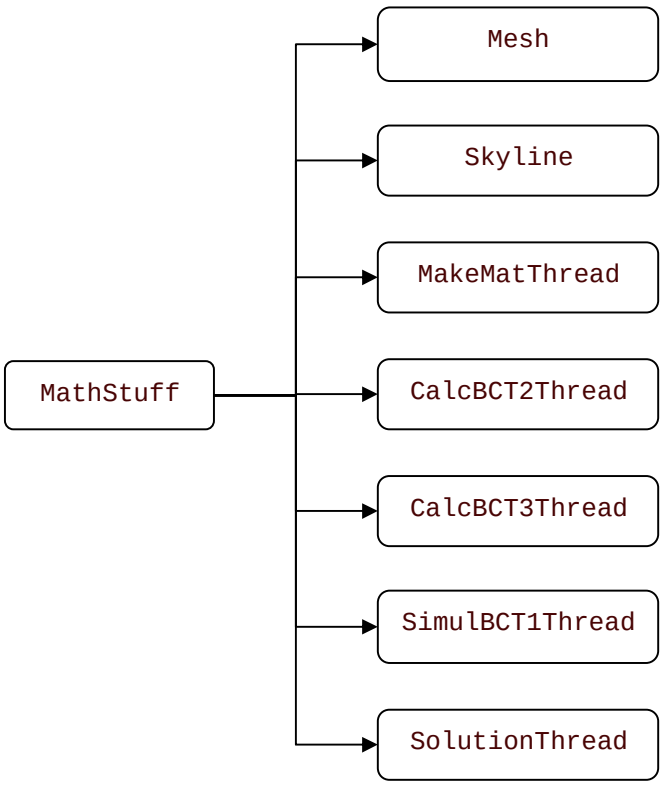
Das Programm besteht aus ungefähr 50 Klassen. Wir haben entschieden, sie in zwei Pakete zu verteilen: ein Paket enthaltet, was die Anzeige betrifft, und heißt „FEJDGui“ und der Andere, „FEJDMath“, umfasst den mathematischen Teil des Programms. FEJavaDemo ist eine Klasse, die unabhängig von diesen zwei Paketen ist. Man benutzt nur FEJavaDemo, um die Software zu starten.
Jetzt werden wir jede Klasse beschreiben und die Links zwischen ihnen darlegen.

2.3.1. Die Klasse „FEJavaDemo“



Die Klasse FEJavaDemo initialisiert das Programm, indem sie Instanzen der Klasse „MathStuff“ und „Display“ schafft. Diese zwei Klassen sind die Hauptklassen des Programms. Um zu vereinfachen, kann man sagen, dass Display dem „FEJDGui“ Paket und MathStuff dem „FEJDMath“ Paket entsprechen.

2.3.2. Die Klasse „MathStuff“

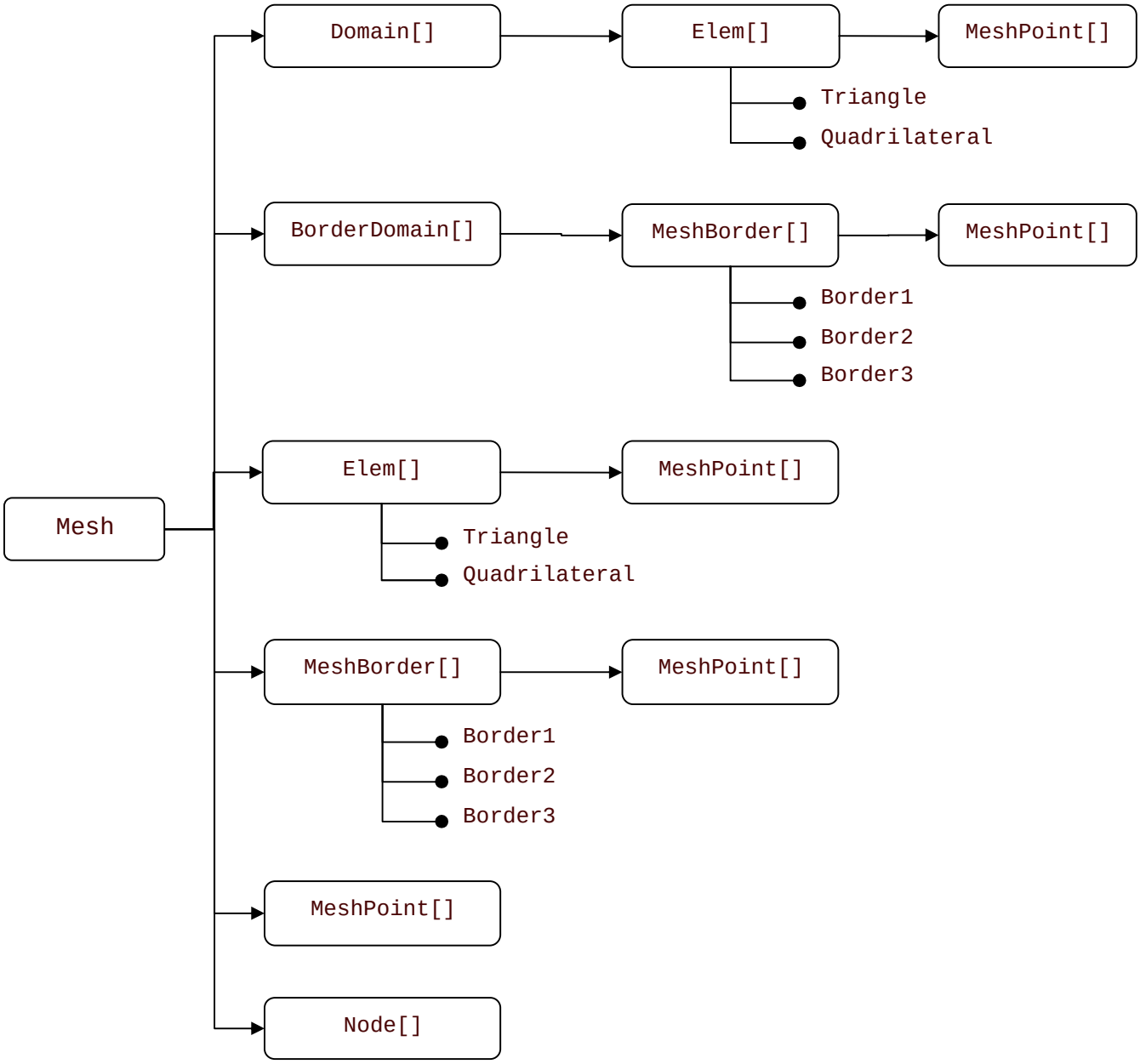


- Die Klasse MathStuff enthält eine Instanz der Klasse „Mesh“, die allen Informationen über die Vernetzung umfasst (siehe unten für eine vollständige Beschreibung).

- MathStuff enthält auch ein „Skyline“ Objekt, das die Steifigkeitsmatrix abspeichert.
  Aufgrund der Wahl der Ansatzfunktionen mit lokalem Träger ist nämlich die Steifigkeitsmatrix schwach besetzt. Die Matrixeinträge K_{ij} sind nur von Null verschieden, wenn der Durchschnitt des Inneren der Träger der Funktionen p_i und p_j nicht leer ist.
  Bei geeigneter Knotennummerierung besitzen dazu die FE-Matrizen eine Bandstruktur. Die Bandweite, d.h. der größte Abstand, den ein Nicht-Null-Element einer Zeile vom entsprechenden Hauptdiagonalelement hat, hängt von der Knotennummerierung ab.
  Diese zwei Eigenschaften soll man bei der Abspeicherung von K ausnutzen. Der Programm benutzt die "Variable Bandweite Spaltenweise (VBS)" oder "Skyline-Speicherung", in dem werden nur die Elemente abgespeichert, die in der Hülle sind, d.h. zwischen einem Nicht-Null-Element einer Zeile und dem entsprechenden Hauptdiagonalelement.

- Schließlich enthält die Klasse MathStuff fünf Threads, die die verschiedenen Schritte des Auflösungsprozesses durchführen. Die benutzten Algorithmen sind unten erklärt.

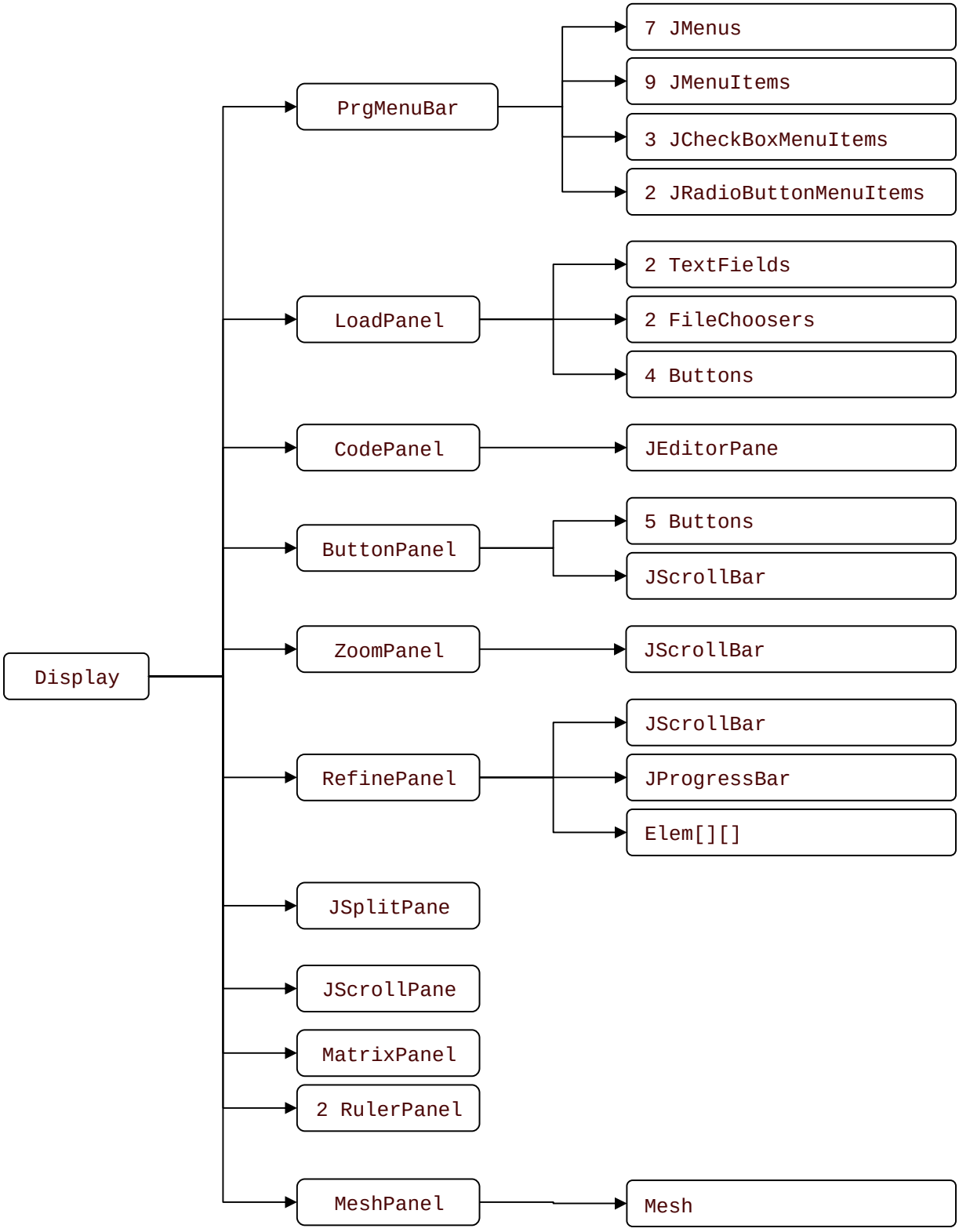
2.3.3. Die Klasse „Mesh“



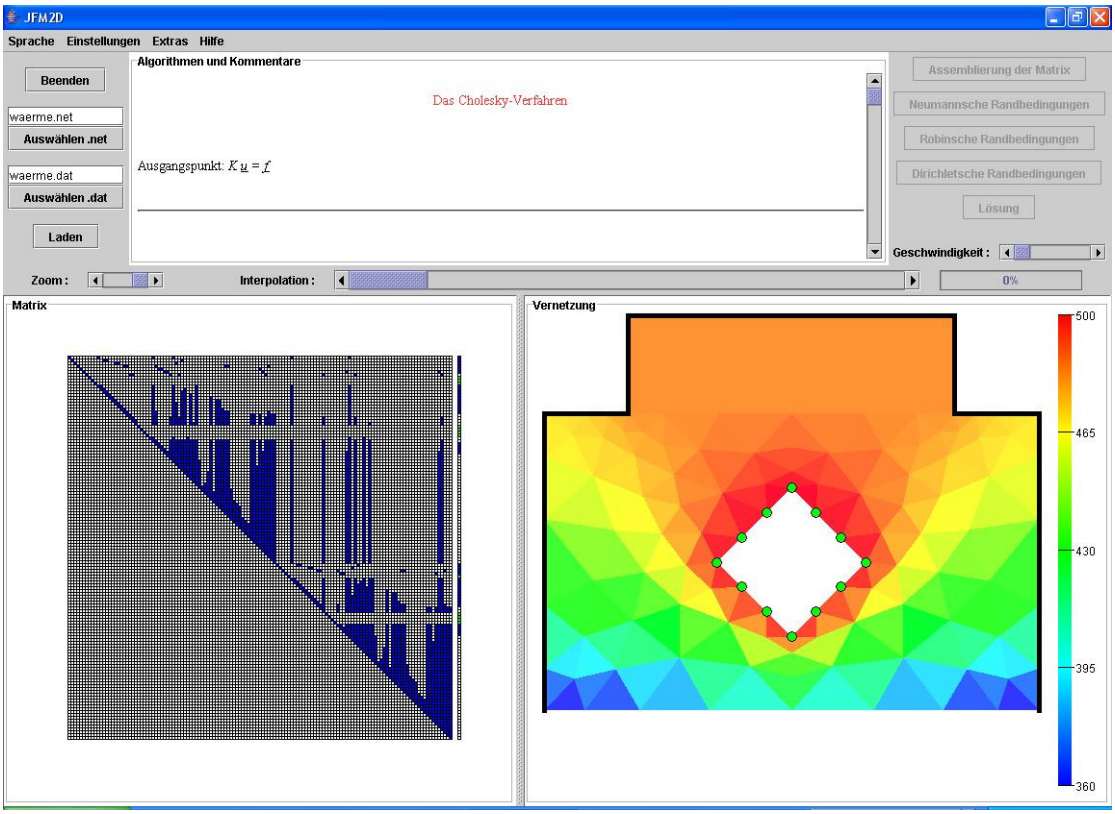
Die Grundklasse, die benutzt wird, um eine Vernetzung zu beschreiben, ist „MeshPoint“.

- Sie darstellt ein Punkt mit seinen zwei Koordinaten, seiner Temperatur, seiner globalen Nummer, usw.
- Wir haben die Knoten („Nodes“) der anderen Punkte („MeshPoints“) differenziert. Während die Knoten nur die Eckpunkte der Elemente sind, sind die „MeshPoints“ alle Punkte, wo man die Temperatur berechnet. Wenn man zum Beispiel mit quadratischen Ansatzfunktionen arbeitet, hat nämlich jedes Dreieck 6 MeshPoints, deren 3 Knoten sind.
  Man brauchte diese Differenzierung, weil man manchmal das Knoten Array braucht (um zum Beispiel die Vernetzung zu zeichnen), und manchmal das MeshPoint Array (um zum Beispiel die Steifigkeitsmatrix zu assemblieren).
- Die Elemente sind mit einem MeshPoint Array definiert. „Elem“ ist eine abstrakte Klasse und die Klassen „Triangle“ und „Quadrilateral“ sind von ihnen abgeleitet. Jedes Element enthält seine Elementsteifigkeitsmatrix und seinen Elementlastvektor, die für die Assemblierung benutzt werden.
- Wir haben eine „Domain“ Klasse geschaffen, die einen Materialbereich darstellt. Jede Klasse enthält ein „Elem“ Array und die Wärmeleitzahlen, die in den Bereich vorgegeben sind.
- Es gibt drei Klassen, die eine Kante darstellen können: „Border1“, „Border2“ und „Border3“. Jede dieser Klassen enthält einen MeshPoint Array, die zwei Knoten und möglicherweise andere MeshPoints umfasst, und die Koeffizienten, die die Randbedingungen definieren. Zum Beispiel gibt es für die „Border3“ zwei Werte: die Wärmeübergangszahl und die Umgebungstemperatur.
- Die Kanten sind in Randbereichen (Klasse „BorderDomain“) zusammengefasst. Wir brauchten so eine Klasse, weil wir die Randbedingungen eine Art nach der anderen einarbeiten.

2.3.4. Die Klasse „Display“



Display ist das Fenster des Programms. Es enthält und initialisiert alle Komponenten der graphischen Oberfläche:



- Die Menüleiste „PrgMenuBar“ besteht aus 4 Menüs. Das „Sprache“ Menü erlaubt dem Benutzer das Programm in Deutsch, in Französisch oder in Englisch zu übersetzen.
  Dank dem „Einstellungen“ Menü ist es möglich die Art der Ansatzfunktionen zu wählen, die benutzt werden, als man die .net und .dat Dateien laden wird. Man kann auch mit diesem Menü die Knotennummern, eine Darstellung der Randbedingungen oder die S^TS Zerlegung anzeigen.
  Der Menüeintrag „Automatische Umnummerierung mit quadratischen Funktionen“ ist standardmäßig ausgewählt. Es ist wichtig, diese Möglichkeit zu benutzen, um den Prozess der Auflösung des Problems schneller zu machen. [siehe Knotenumnummerierung (3.1)].
  Eine letzte Auswahlmöglichkeit dieses Menü kann benutzt werden, um neue Dateien zu erstellen, wenn man eine Umnummerierung der Knoten oder eine Verfeinerung der Vernetzung durchgeführt wird. Das kann nötig sein, um später eine gute Vernetzung zu

laden, ohne dieselben Schritte zu wiederholen.
Das Menü „Extras“ schlägt vor, Jpeg Bilder der Matrix und der Vernetzung zu erstellen, die Lösung in einer Datei zu speichern (d.h. die Knoten mit ihren Koordinaten und Temperaturen), die Knoten umzunummerieren und die Vernetzung zu verfeinern. Für weitere Informationen über diese zwei letzten Möglichkeiten, siehe drittes Kapitel.

- Wenn das Programm anläuft, soll der Benutzer die zwei Dateien laden, die unter anderem die Vernetzung beschreiben [Siehe Anlage 2 – Struktur der Eingabefiles]. Das „LoadPanel“ bietet die Möglichkeit, diese Dateien von der Festplatte auszuwählen. Zweitrangig gibt es auch ein Knopf, um FEJavaDemo zu beenden.

- Das „CodePanel“ zeigt die verschieden Algorithmen, die genutzt werden, um die Problembearbeitung durchzuführen. Auf diese Weise kann der Benutzer sich über die verwendeten Methoden einen Überblick verschaffen, ohne den Code lesen zu müssen. So sieht man die Verbindung zwischen der Theorie und der Anzeige des Programms (zum Beispiel kann der Benutzer verstehen, warum solche Elemente eingefärbt sind).

- Das „ButtonPanel“ wird dazu benutzt, die Problembearbeitung durchzuführen. Ein Knopf löst die Assemblierung der globalen Steifigkeitsmatrix aus und drei andere erlauben, die Randbedingungen zu berücksichtigen. Ein letzter Knopf leitet den Auflösungsprozess ein. Siehe unten für weitere Details über die genutzten Algorithmen. Ein Scroll bar ändert die Geschwindigkeit der Anzeige, um dem Ablauf der Prozesse folgen zu können. Oft zieht man vor, dass das Programm schnell die Lösung liefert. Deshalb sieht man die Stufen der Rechnung nicht mehr, wenn der Rollbalken am weitesten Links ist.

- Das „MatrixPanel“ ist eines des wichtigen Panels von FEJavaDemo. Man kann nämlich auf ihm die globale Steifigkeitsmatrix und den globalen Lastvektor sehen. Da die Dimension der Matrix oft zu groß ist für die Oberfläche, hat man das Panel in einem Scrollpane gelegt, der dazu dient, in diesem Fall horizontale und vertikale Scrollbar hinzuzufügen. Zwei Leisten deuten die Nummern der Spalten und Zeilen an. Die Nicht-Null-Einträge sind eingefärbt, um hervorzuheben, dass die Matrix schwach besetzt ist. Während den verschiedenen Assemblierungsprozesse sind die Elemente, mit denen das Programm arbeitet, in rot ausgemalt. Die grünen Elemente entsprechen den Dirichlet-Knoten. Wenn das adäquate Ankreuzfeld ausgewählt ist, kann man auch die obere Dreiecksmatrix S der

S^TS Zerlegung sehen. Ein kleiner Zeichentrick stellt die Durchführung des Vorwärts- und Rückwärtseinsetzen dar.

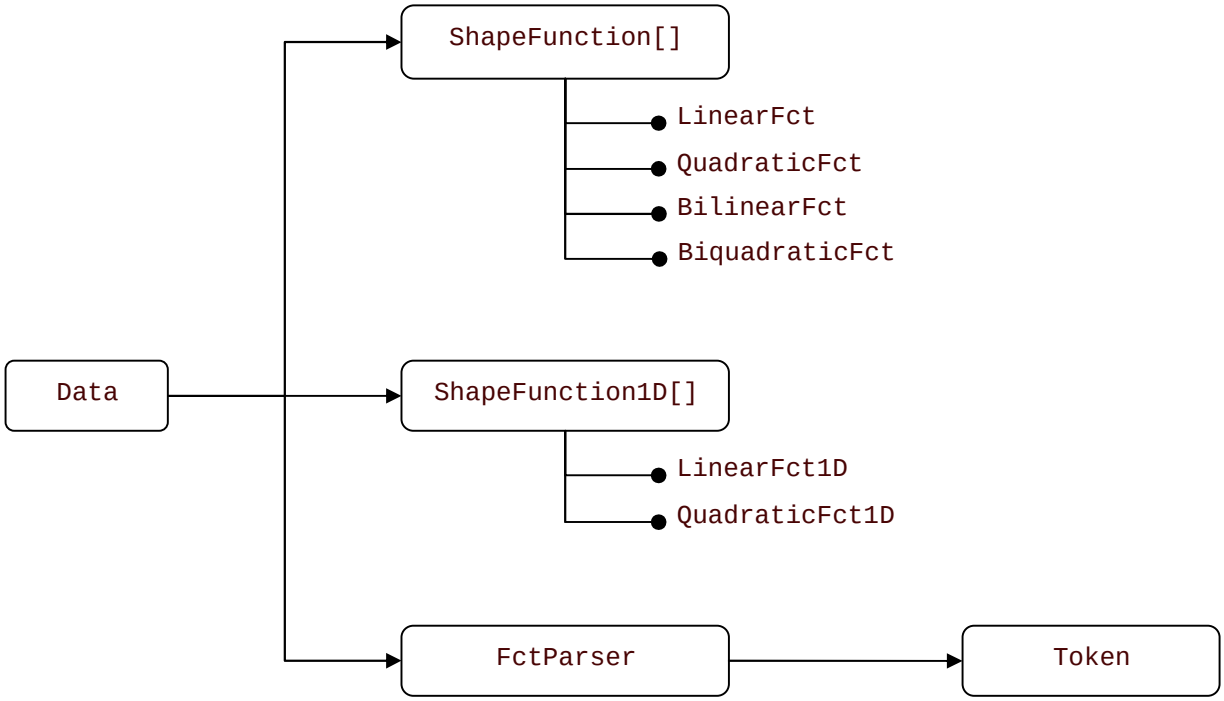
- Das „ZoomPanel“ enthält einen Scroll bar, den des Zoom des MatrixPanel ändert. Wenn der Rollbalken am weitesten Links ist, sieht man die Werte der Einträge.

- Auf dem „MeshPanel“ kann man die Vernetzung sehen. Wenn die Auflösung fertig ist, ist das Temperaturfeld neben einer Temperaturskala dargestellt. Wie für das MatrixPanel sind die Elemente, mit der das Programm arbeitet, in rot ausgemalt. So kann man die zwei Panels gleichsetzen, um das Prinzip der Assemblierung zu verstehen. Dazu kann man dank dem Menü „Einstellungen“ die Knotennummern anzeigen und eine Darstellung der Randbedingungen. Das heißt, dass die Dirichlet-Knoten in Grün eingefärbt und die wärmeisolierten Kanten betont sind.

- Das „RefinePanel“ erlaubt die Lösungsdarstellung zu verbessern. Die Dreiecke können intern wiederholt in vier Teildreiecke zerlegt werden und die Lösung wird über ihnen linear interpoliert. Da es ziemlich lang dauern kann, haben wir einen Fortschrittsbalken (Progress Bar) hinzugefügt, der dazu dient, dem Benutzer ein Feedback über den aktuellen Status der Arbeit zu geben.

- Wir haben ein Splitpanel implementiert, das die Möglichkeit bietet, zwei Komponenten nebeneinander anzuzeigen, wobei bei Vergrößerung des Anzeigebereichs der einen Komponente der sichtbare Bereich der anderen proportional abnimmt. Was das Programm betrifft war es interessant, damit der Benutzer die Matrix oder die Vernetzung größer sehen kann.

2.3.5. Die Klasse „Data“

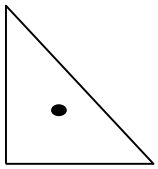
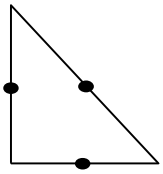
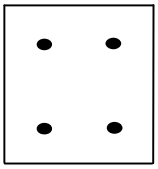
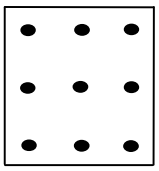


„Data“ ist eine unabhängige Klasse, die genutzt wird, um die Elementsteifigkeitsmatrizen und die Elementlastvektoren zu berechnen. Sie enthält die Koeffizienten der Ansatzfunktionen und die Methoden, um Integrale zu berechnen.

Bei der Berechnung der Einträge der Matrizen und Vektoren ist es nicht immer möglich, die entsprechenden Integrale analytisch zu berechnen. Deshalb nutzt das Programm Quadraturformeln zur Berechnung der Integrale. Die Grundidee der Quadraturformeln besteht darin, die Integrale durch Summen zu approximieren. Die Software benutzt die Quadratur der Gestalt:

$$\int_{\bar{T}} w(\xi) d\xi \approx \sum_{i=1}^{l} \alpha_i w(\xi_i)$$

Da die Polynome, die das Programm integrieren muss, immer von relativ kleinem Grad sind, kann FEJavaDemo Quadraturstützstellen benutzen, um diese exakt zu integrieren:

Lage der Stützstellen	Quadraturstützstellen ξ_i oder $(\xi_{i,1}, \xi_{i,2})$	Gewichte α_i	exakt für Polynome vom Grad k
	$\frac{3-\sqrt{3}}{6}, \frac{3+\sqrt{3}}{6}$	$\frac{1}{2}, \frac{1}{2}$	k = 3
	$\frac{5-\sqrt{15}}{10}, \frac{1}{2}, \frac{5+\sqrt{15}}{10}$	$\frac{5}{18}, \frac{8}{18}, \frac{5}{18}$	k = 5
	$\left(\frac{1}{3}, \frac{1}{3}\right)$	$\frac{1}{2}$	k = 1
	$\left(\frac{1}{2}, 0\right), \left(\frac{1}{2}, \frac{1}{2}\right), \left(0, \frac{1}{2}\right)$	$\frac{1}{6}, \frac{1}{6}, \frac{1}{6}$	k = 2
	$\left(\frac{3-\sqrt{3}}{6}, \frac{3+\sqrt{3}}{6}\right), \left(\frac{3+\sqrt{3}}{6}, \frac{3+\sqrt{3}}{6}\right)$ $\left(\frac{3-\sqrt{3}}{6}, \frac{3-\sqrt{3}}{6}\right), \left(\frac{3+\sqrt{3}}{6}, \frac{3-\sqrt{3}}{6}\right)$	$\frac{1}{4}, \frac{1}{4}$ $\frac{1}{4}, \frac{1}{4}$	k = 3
	$\left(\frac{5-\sqrt{15}}{10}, \frac{5+\sqrt{15}}{10}\right)\left(\frac{1}{2}, \frac{5+\sqrt{15}}{10}\right)\left(\frac{5+\sqrt{15}}{10}, \frac{5+\sqrt{15}}{10}\right)$ $\left(\frac{5-\sqrt{15}}{10}, \frac{1}{2}\right)\left(\frac{1}{2}, \frac{1}{2}\right)\left(\frac{5+\sqrt{15}}{10}, \frac{1}{2}\right)$ $\left(\frac{5-\sqrt{15}}{10}, \frac{5-\sqrt{15}}{10}\right)\left(\frac{1}{2}, \frac{5-\sqrt{15}}{10}\right)\left(\frac{5+\sqrt{15}}{10}, \frac{5-\sqrt{15}}{10}\right)$	$\frac{25}{324}, \frac{10}{81}, \frac{25}{324}$ $\frac{10}{81}, \frac{16}{81}, \frac{10}{81}$ $\frac{25}{324}, \frac{10}{81}, \frac{25}{324}$	k = 5

Quadraturformeln über dem Referenzintervall I = [0,1] und über dem Referenzelement

Bemerkung: Bei Vierecken sind die Formeln exakt für Funktionen, die Polynome k-ten Grades in jeder Raumrichtung sind.

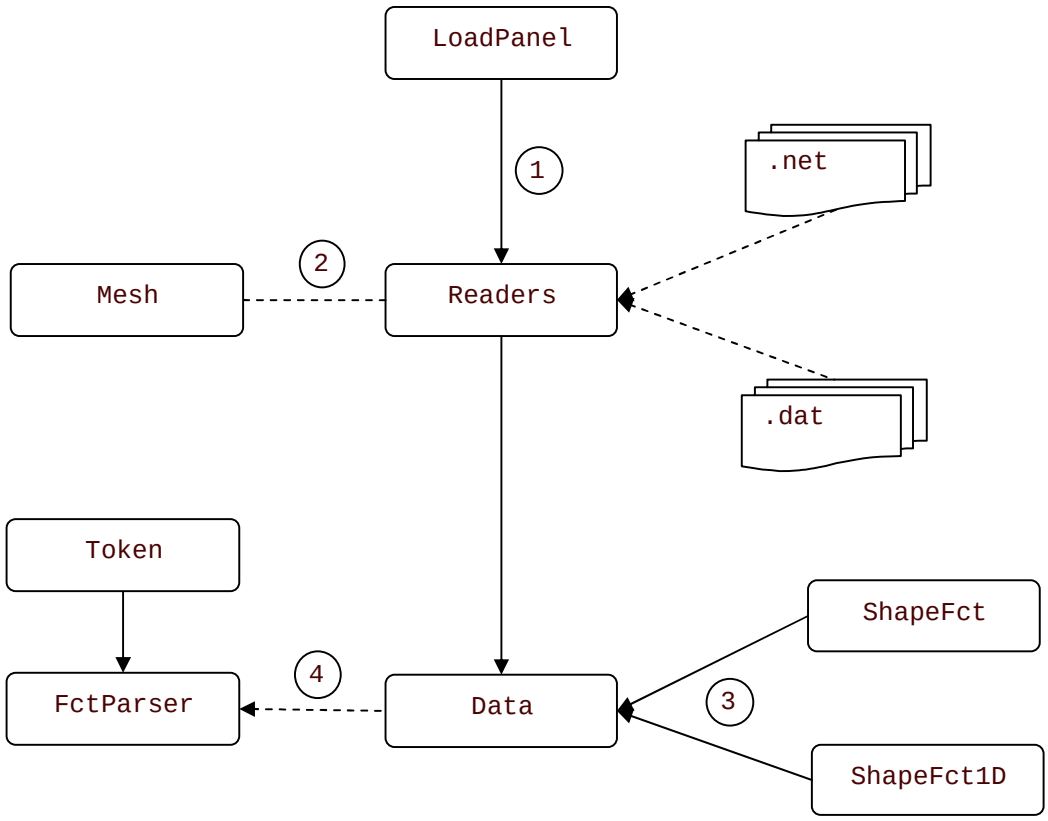
Die numerische Integration wird mehrfach genutzt:

- Bei der Berechnung der Elementsteifigkeitsmatrizen und der Elementlastvektoren. Das Programm führt in den Integralen eine Variablensubstitution durch, so dass Integrale nur über dem Referenzelement (Dreieck, Viereck, usw.) zu berechnen sind.
- Bei der Einarbeitung der Randbedingungen, rechnet FEJavaDemo auf analoge Weise nur über dem Referenzintervall I = [0,1].

2.4. Prozesse

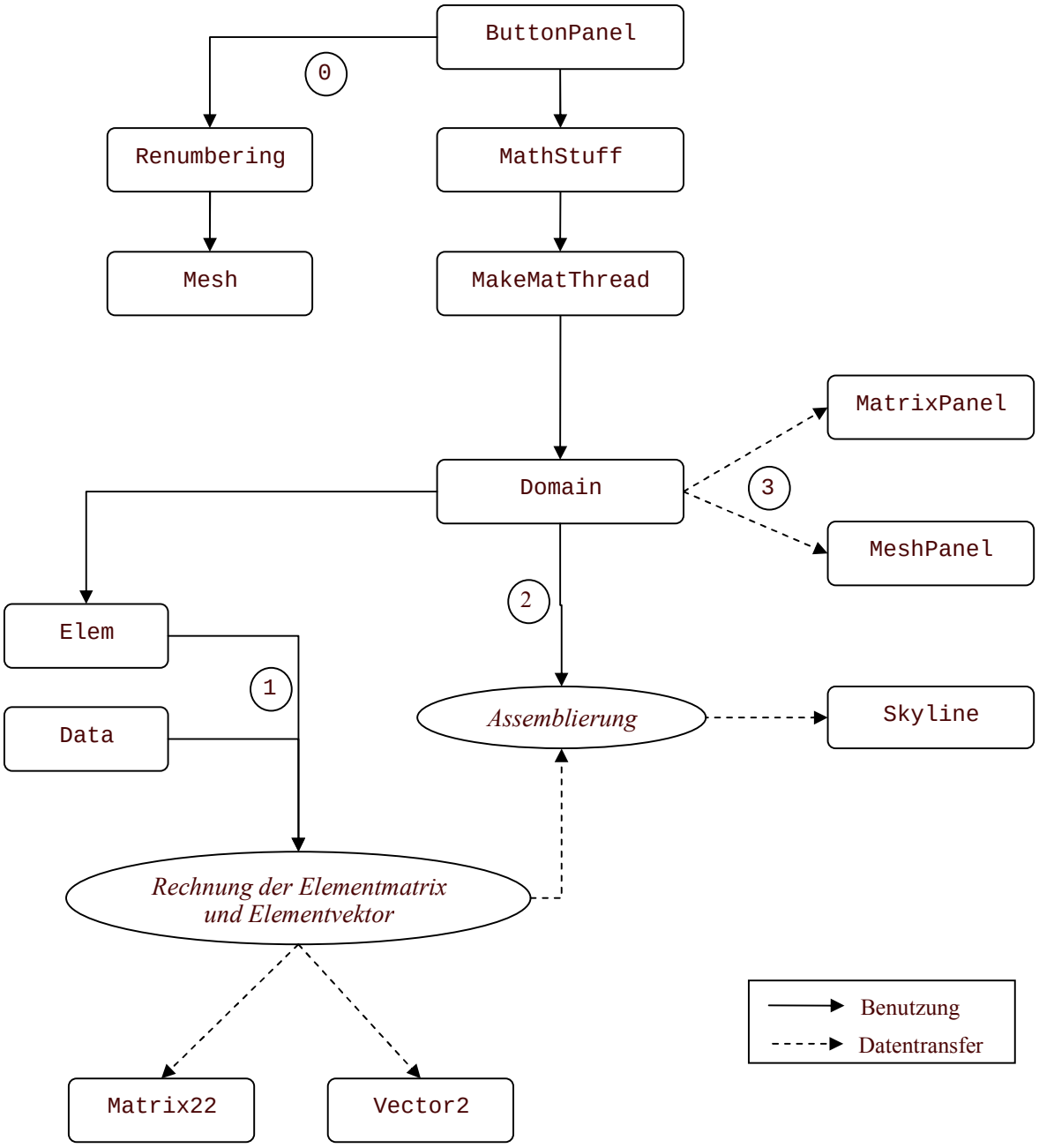
In diesem Kapitel werden die Wechselwirkungen zwischen den verschiedenen Klassen beschrieben. Für jeden Prozess kann man sehen, welche Klassen betroffen sind und die Reihenfolge der Schritte. Die Algorithmen, die in Ovalen sind, werden in dem nächsten Kapitel erklärt.

2.4.1. Laden der Dateien



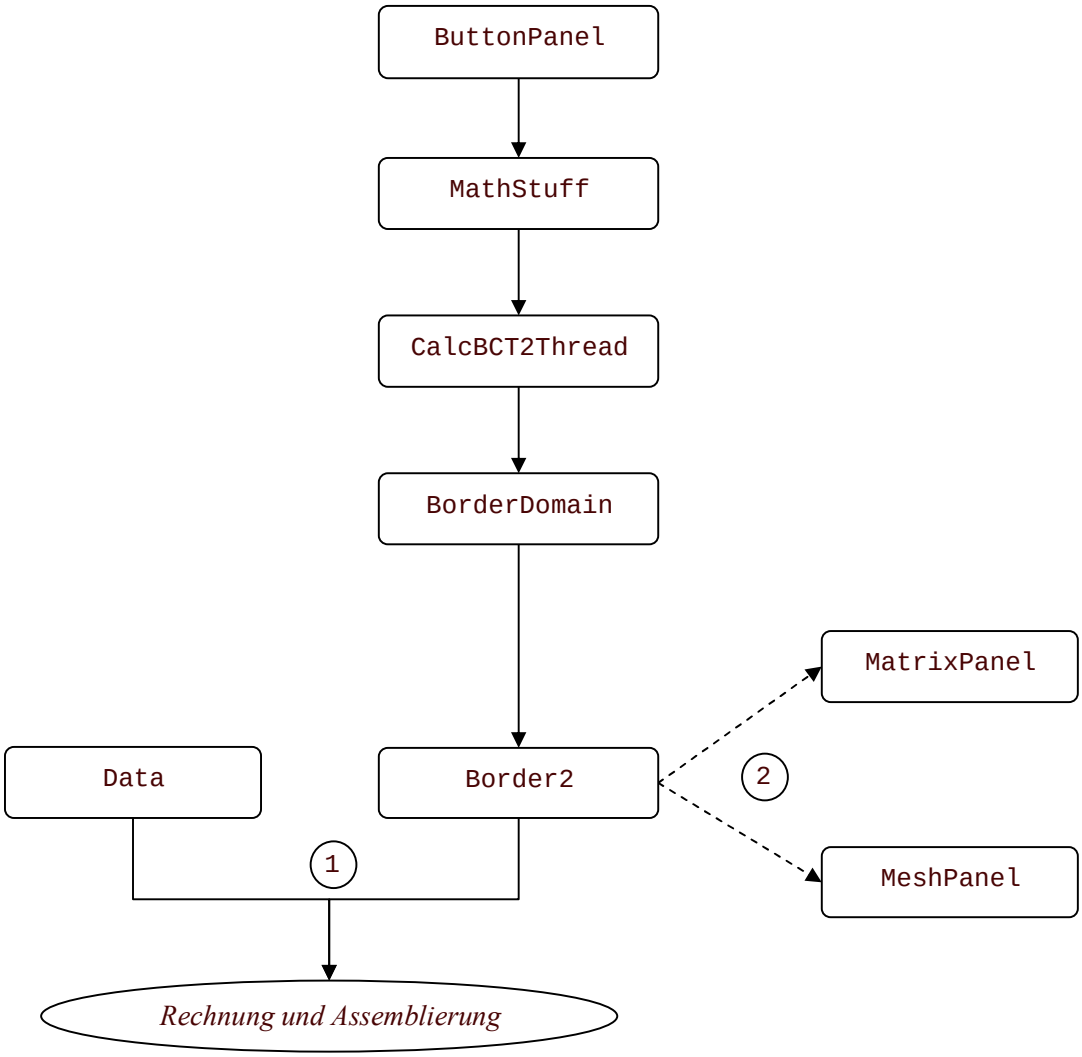
1) LP benutzt ein „Readers“-Objekt, um die Dateien zu lesen
2) „Readers“ füllt das „Mesh“-Objekt mit den Informationen der Dateien.
3) „Readers“ ruft das „Data“-Objekt, damit es dank „ShapeFct“ und „ShapeFct1D“ die Tabelle der Werte der Ansatzfunktionen in jeder Quadraturstützstelle füllt.
4) Data gibt „FctParser“ die f Funktion als „String“-Objekt. Das „FctParser“-Objekt benutzt dann die „Token“-Objekte, um sie in Token auszuwachsen, damit sie erlernbar für das Programm werden.

2.4.2. Assemblierung der Matrix



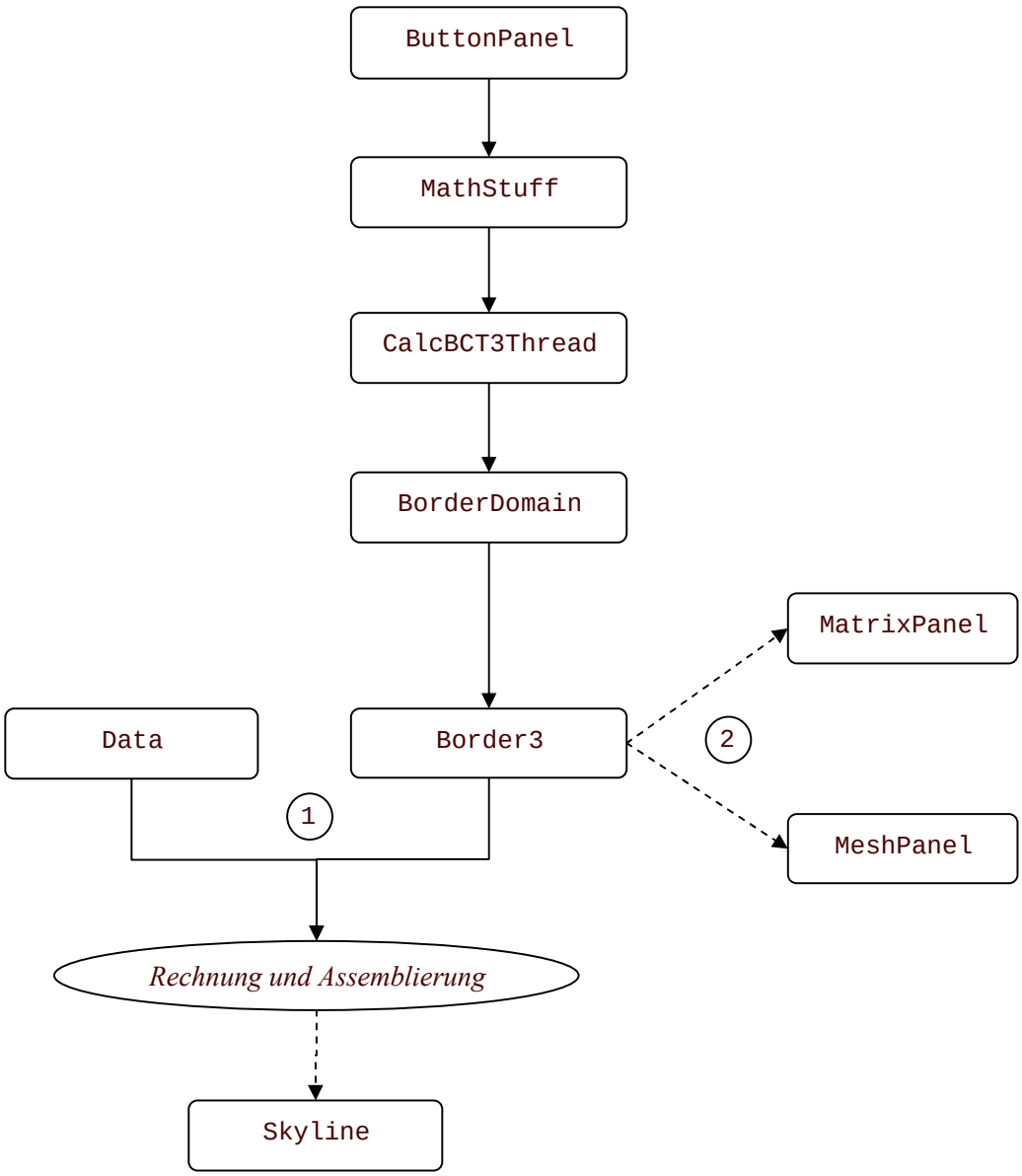
1) Notfalls (mit quadratischen Ansatzfunktionen) wird die Vernetzung „Mesh“ umnummeriert.
2) Eine Schleife über alle Berichte ist nötig, um die verschiedenen Wärmeleitzahlen berücksichtigen. Dann rechnet das Programm auf Elementebene die Elementsteifigkeitsmatrizen und Elementlastvektoren aus, die in „Vector2“ und „Matrix22“ abgespeichert werden. Die Integrale werden dank dem „Data“-Objekt berechnet.
3) Nach der Bildung einer Elementmatrix und Elementvektor, ruft das „Domain“-Objekt (Bereich) die Assemblierung und füllt die globale Steifigkeitsmatrix, die in einem „Skyline“-Objekt abgespeichert wird.
4) Für jedes Element informiert das „Domain“-Objekt das „MeshPanel“ und das „MatrixPanel“ über das bearbeitete Element, damit es in Rot eingefärbt werden kann.

2.4.3. Berücksichtigung der Randbedingungen 2. Art



1) Nach einer Schleife über den Randbereichen, die Kanten 2. Art enthalten (gewöhnlich gibt es nur einen solchen Randbereich) assembliert das Programm auf Ebene der Kanten die Elementvektoren
2) Für jede Neumannsche Kante informiert das „Border2"-Objekt das „MeshPanel" und das „MatrixPanel" über die bearbeitete Kante, damit sie in Rot eingefärbt werden kann

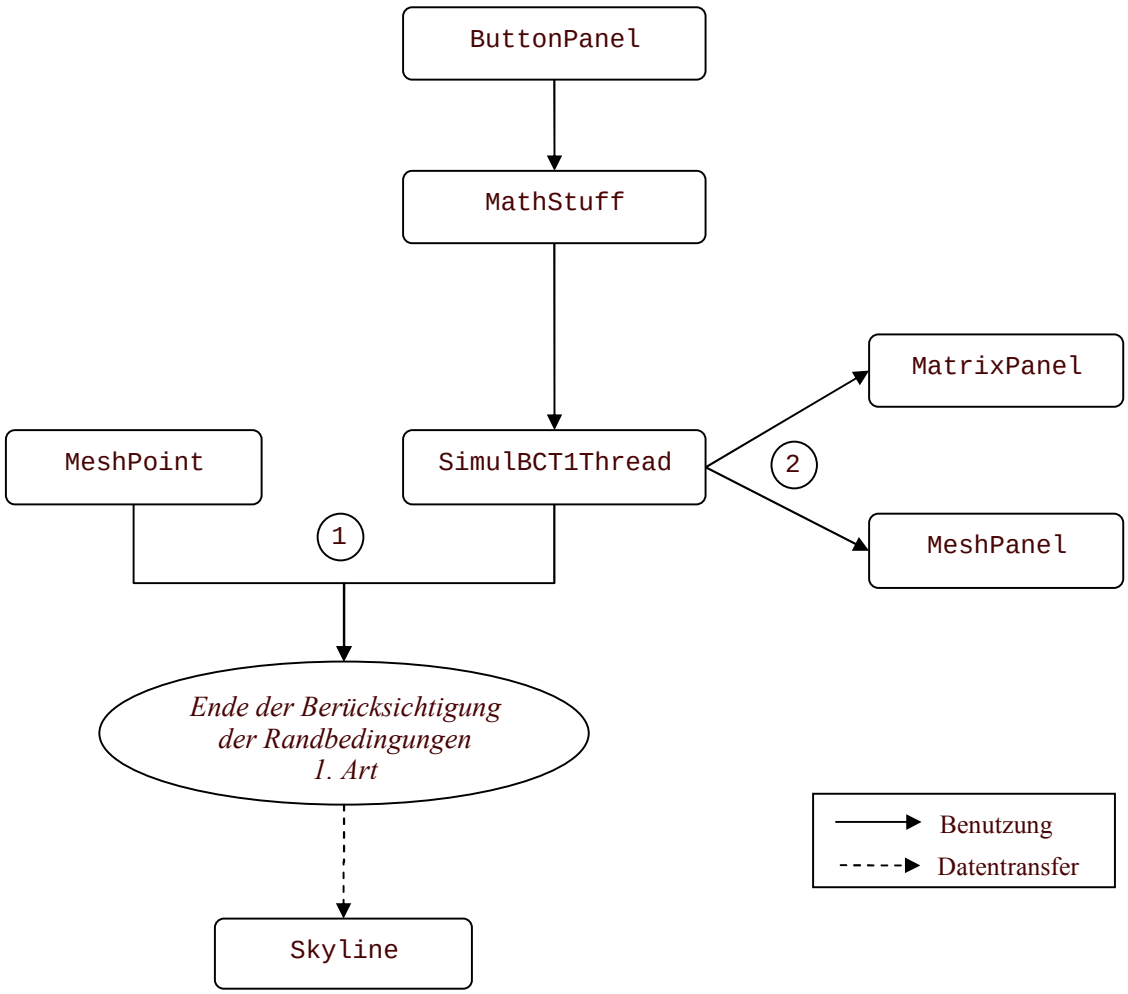
2.4.4. Berücksichtigung der Randbedingungen 3. Art



1) Nach einer Schleife über den Randbereichen, die Kanten 3. Art enthalten (gewöhnlich gibt es nur einen solchen Randbereich) assembliert das Programm auf Ebene der Kanten die Elementvektoren und Elementmatrizen, die die Skyline ändern.

2) Für jede Robinsche Kante informiert das „Border3“-Objekt das „MeshPanel“ und das „MatrixPanel“ über die bearbeitete Kante, damit sie im Rot eingefärbt werden kann.

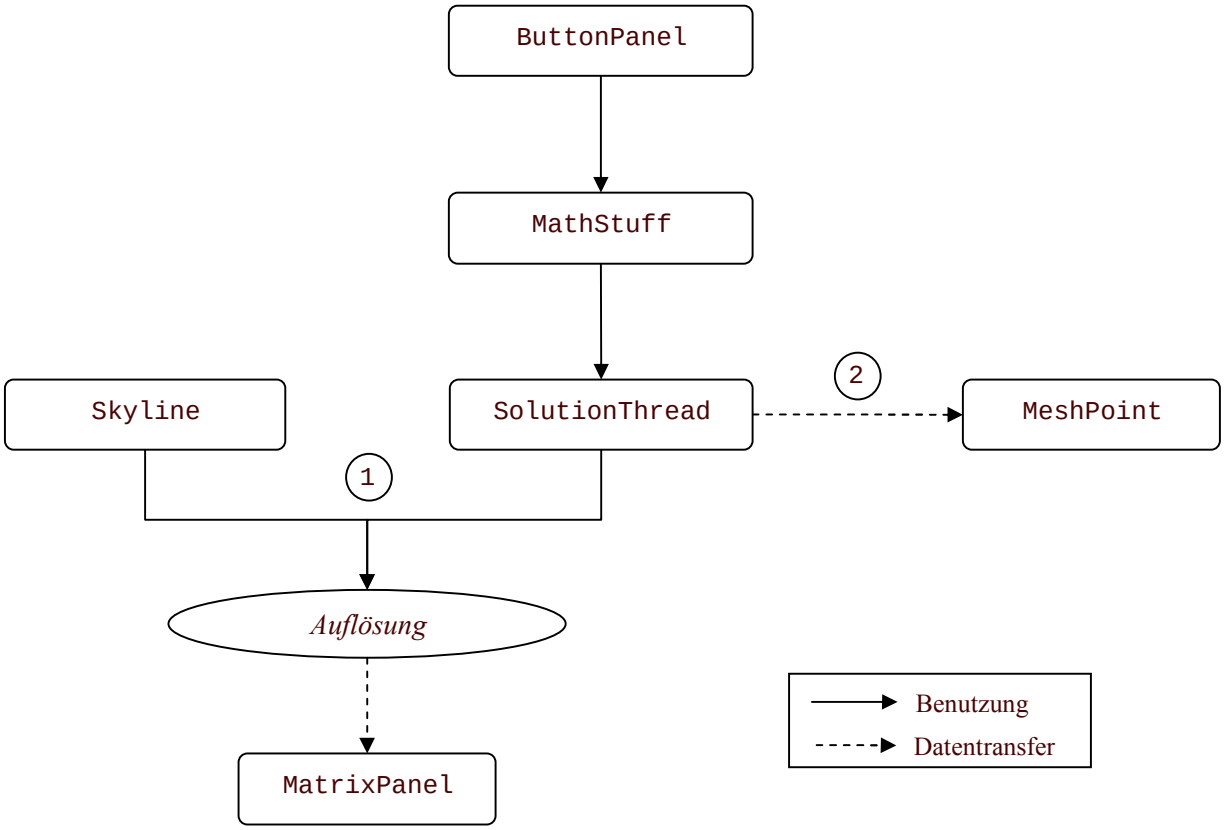
2.4.5. Berücksichtigung der Randbedingungen 1. Art



1) Die Berücksichtigung der Randbedingungen 1. Art wird zum Teil während des Assemblierungsprozesses und der Berücksichtigung der Randbedingungen 3. Art behandelt. Jetzt sind nur noch, „1“ auf die Diagonale der globalen Steifigkeitsmatrix und die Werte der Dirichlet-Knoten auf den Lastvektor zu speichern. Da die Dirichletsche Bedingung nicht auf Kanten sondern auf Knoten definiert sind, macht das Programm die Schleife auf Ebene der Knoten der Vernetzungen.

2) Für jede Dirichletschen Knoten informiert das „SimulBCT1Thread“-Objekt das „MeshPanel“ und das „MatrixPanel“ über der bearbeitete Knoten, damit er in Grün eingefärbt werden kann.

2.4.6. Auflösung des Problems



1) Zuerst ruft die Thread mit dem „Skyline“-Objekt den Algorithmus der Auflösung des Problems. Das „Skyline“-Objekt enthält die Steifigkeitsmatrix und einige Methoden, um sie zu benutzen: Cholesky-Zerlegung, Lösung von linearen Gleichungssysteme, usw. Dann wird das „MatrixPanel“ gerufen, um die obere Dreiecksmatrix S der S^TS Zerlegung und der kleine Zeichentrick, der die Durchführung des Vorwärts- und Rückwärtseinsetzen symbolisiert, zu zeigen.

2) Die Temperaturwerte werden den „MeshPoint“ gegeben.

2.5. Algorithmen

2.5.1. Berechnung und Assemblierung der Steifigkeitsmatrix und des Lastvektors

Der Aufbau des FE-Gleichungssytems wird elementweise durchgeführt. Zuerst assembliert das Programm die Steifigkeitsmatrix und den Lastvektor ohne Berücksichtigung der Randbedingungen.

Bei der Berechnung der Elementsteifigkeitsmatrizen $K^{(r)}$ führen wir in den Integralen eine Variablensubstitution durch, so dass Integrale über dem Referenzelement $\hat{T}$ zu berechnen sind.
Für das Elements $T^{(r)}$ erhalten wir die Elementsteifigkeitsmatrix $K^{(r)}$:

$$K^{(r)} = \left[\int_{\hat{T}} \left[\left(grad_{\xi} \varphi_i(\xi) \right)^T \left(J^{(r)} \right)^{-1} \Lambda \left(J^{(r)} \right)^{-T} grad_{\xi} \varphi_j(\xi) \right] \left| \det J^{(r)} \right| d\xi \right]_{i,j=1}^{\hat{N}}$$

mit:
- ξ : Koordinate über dem Referenzelement
- φ_i und φ_j sind die auf dem Referenzelement definierten Formfunktionen
- $\Lambda = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}$: Wärmeleitzahlen des Materials
- $\hat{N}$: Anzahl der Knoten pro Element
- $J^{(r)}$: Jacobi-Matrix der Transformation, die die Abbildung des Referenzelements $\hat{T}$ auf ein beliebiges Element $T^{(r)}$ der Vernetzung realisiert.

Nur die Matrizen $J^{(r)}$, $(J^{(r)})^{-1}$ und die Ableitungen der über dem Referenzelement definierten Formfunktionen werden bei der Berechnung der Einträge der Elementsteifigkeitsmatrizen benötigt. Explizite Formeln für die Formfunktionen über den Elementen $T^{(r)}$ sind nicht erforderlich.

Die Elementlastvektoren $f^{(r)}$ werden auf analoge berechnet, d.h.

$$\underline{f}^{(r)} = \left[\int_{\bar{T}} f\left(x_{T^{(r)}}(\xi)\right) \varphi_i(\xi) \left| \det J^{(r)} \right| \mathrm{d}\xi \right]_{i=1}^{\bar{N}}$$

mit: $x_{T^{(r)}}(\xi) = J^{(r)}\xi + x_1^{(r)}$: Koordinaten über dem Element $T^{(r)}$

Während dieses ersten Prozesses werden auch die Steifigkeitsmatrix und der Lastvektor modifiziert, um die Randbedingungen 1. Art zu berücksichtigen.

Die Steifigkeitsmatrix wird durch

$$K_{ij}^{(r)} = K_{ji}^{(r)} = \delta_{ij} \qquad \forall i \in \gamma_h^{(r)}, j \in \bar{\omega}_h^{(r)}$$

korrigiert, mit $\bar{\omega}_h^{(r)}$: Indexmenge, welche die Nummern der Knoten im Element $T^{(r)}$ enthält.

Wir korrigieren auch die rechte Seite durch:

$$f_i^{(r)} := f_i^{(r)} - \sum_{j \in \gamma_h^{(r)}} K_{ij} g_1(x_j) \qquad \forall i \in \omega_h^{(r)}$$

mit:

- $g_1(x_j)$: Wert der Dirichlet Randbedingung im Punkt x_j
- $\omega_h^{(r)}$: Indexmenge, welche die Nummern der Nicht-Dirichlet-Knoten im Element $T^{(r)}$ enthält.
- $\gamma_h^{(r)}$: Indexmenge, welche die Nummern der Dirichlet-Knoten im Element $T^{(r)}$ enthält.

Wir könnten auch diese Korrekturen am Ende des Assemblierungsprozesses machen. Allerdings ist es bei der Skyline-Speichertechnik für die Matrix aufwendig, die Matrixeinträge der Zeilen zu finden, die genullt werden müssen.

Dann werden die Elementsteifigkeitsmatrizen und Elementlastvektoren durch den folgenden Algorithmus assembliert, der eine Schleife über die Elementbereiche macht, weil verschiedene Bereiche verschiedene Wärmeleitzahlen haben können.

Für jeden Elementbereich

Für jedes Element $T^{(r)}$ des Bereichs

Berechne $K^{(r)}$ und $\underline{f}^{(r)}$

Berücksichtigung der Randedingungen 1. Art: $\underline{f}^{(r)}$ und $K^{(r)}$ werden korrigiert

Für jeden Knoten des Elements, der i als globale Nummer und k als lokale Nummer hat.

$$f_i := f_i + f_k^{(r)}$$

Für Jeden Knoten des Elements, der j als globale Nummer und l als lokale Nummer hat.

Der nächste Schritt ist die Berücksichtigung der Randbedingungen:

Randbedingungen 2. Art:

Vektor, der für die Neumannschen Randbedingungen assembliert werden muss:

$$\underline{f}^{(e_2)} = \left[\left(\int_0^1 \varphi_i(\xi) d\xi \right) g_2 \sqrt{\left(x_{n_2} - x_{n_1}\right)^2 + \left(y_{n_2} - y_{n_1}\right)^2} \right]_{i=1}^{\hat{N}_E}$$

mit:

- φ_i sind die über dem Referenzintervall definierten Formfunktionen
- $\hat{N}_E$: Anzahl der Knoten pro Kante
- n_1 und n_2 : Anfangs- und Endknote der Kante
- g_2 : Wert der Randbedingungen auf der Kante

Randbedingungen 3. Art:

Vektor, der für die Robinschen Randbedingungen assembliert werden muss:

$$\underline{f}^{(e_3)} = \left[\left(\int_0^1 \varphi_i(\xi) d\xi \right) \alpha \, u_A \sqrt{\left(x_{n_2} - x_{n_1}\right)^2 + \left(y_{n_2} - y_{n_1}\right)^2} \right]_{i=1}^{\tilde{N}_E}$$

mit: α und u_A : Werte der Randbedingungen auf der Kante

Matrix, die für die Robinschen Randbedingungen assembliert werden muss:

$$K^{(e_3)} = \left[\left(\int_0^1 \varphi_i(\xi) \varphi_j(\xi) d\xi \right) \alpha \sqrt{\left(x_{n_2} - x_{n_1}\right)^2 + \left(y_{n_2} - y_{n_1}\right)^2} \right]_{i=1}^{\tilde{N}_E}$$

Natürlich korrigiert das Programm auch den Vektor und die Matrix, die Robinschen Randbedingungen entsprechend, um die Randbedingungen 1. Art zu berücksichtigen. Sie werden analog wie die Elementsteifigkeitsmatrix und der Elementlastvektor korrigiert.

Diese Matrizen und Vektoren werden mit denselben Algorithmus wie die Elementsteifigkeitsmatrizen und Elementvektoren assembliert.

Randbedingungen 1. Art:

Da die Berücksichtigung der Randbedingungen 1. Art während der Berechnung der Steifigkeitsmatrix und des Lastvektors ohne Berücksichtigung der Randbedingungen und während der Einarbeitung der Robinschen Randbedingungen behandelt wird, muss jetzt nur noch „1“ auf die Diagonale der globalen Steifigkeitsmatrix und die Werte der Dirichlet-Knoten auf den Lastvektor gespeichert werden.

Für jeden Dirichlet-Knoten des Elements, der i als globale Nummer hat,

$K_{ii} = 1$

$f_i = g_1(x_i)$

mit $g_1(x_i)$: Wert der Dirichletschen Randbedingung im Punkt x_i.

2.5.2. Lösung des FE-Gleichungssystems

Als Resultat der FE-Diskretisierung haben wir ein lineares Gleichungssystem: $K\,u = f$.
K ist die Steifigkeitsmatrix, f der Lastvektor und u der Lösungsvektor, den wir suchen.

Zur Lösung der FE-Gleichungssysteme kann man sowohl direkte als auch iterative Verfahren benutzen. Direkte Auflösungsverfahren sind Algorithmen, die den Lösungsvektor u in endlich vielen Schritten liefern. Iterative Verfahren bestimmen den Vektor u ausgehend von einer Startnäherung u_0 als Grenzwert einer Folge von Näherungslösungen (u_k).

Wir wissen, dass die Matrix K symmetrisch und positiv definiert ist und in diesem Fall wird häufig das *Cholesky-Verfahren* eingesetzt, das ein direktes Verfahren ist.

Beim Cholesky-Verfahren wird die Matrix K in das Produkt einer oberen Dreiecksmatrix und ihrer transponierte Matrix zerlegt: $K = S^T S$

Wir wissen auch, dass die Matrix K schwach besetzt ist. Bei der Durchführung der Faktorisierung bleiben gewisse Besetztheitsstrukturen der Matrix K erhalten. So kann man die Matrizen K und S auf den gleichen Speicherplätzen abspeichern. Und vor allem kann das Programm einen besonderen Zerlegungsalgorithmus benutzen, der die Nulleinträge außerhalb der Hülle nicht berechnet:

Zerlegungsalgorithmus ($K = S^T S$) mit Berücksichtigung der Profilstruktur von K:

$S_{11} = \sqrt{K_{11}}$

Berechne für $j = 2, 3, \ldots, N$

Falls $l_0(j)+1 < j$, berechne für i = $l_0(j)+1, l_0(j)+2, \ldots, j-1$:

$$S_{ij} = \frac{1}{S_{ii}}\left(K_{ij} - \sum_{l=\max\{l_0(i),l_0(j)\}+1}^{i-1} S_{li} S_{lj} \right)$$

$$S_{jj} = \sqrt{K_{jj} - \sum_{l=l_0(j)+1}^{j-1} S_{lj}^2}$$

(Dabei bezeichnet $l_0(j)$ den Zeilenindex, für den in der j-ten Spalte $K_{ij} = 0$ für alle $0 < i < l_0(j) + 1$ und $K_{l_0(j)+1,\,j} \neq 0$ gilt)

Die Lösung von $Ku = f$ ist somit der Lösung von $S^T S\, u = f$ äquivalent. Dieses Gleichungssystem löst das Programm in zwei Etappen durch *Vorwärts-* und *Rückwärtseinsetzen*, d.h. es löst zunächst $S^T y = f$ und dann $S\, u = y$. Die Systemmatrizen dieser zwei Gleichungssysteme haben eine Dreiecksgestalt. Derartige Gleichungssysteme sind besonders einfach lösbar:

Algorithmus des Vorwärtseinsetzen:

$$\begin{pmatrix} S_{11} & 0 & \cdots & 0 \\ S_{12} & S_{22} & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ S_{1N} & S_{2N} & \cdots & S_{NN} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_N \end{pmatrix}$$

$$y_1 = \frac{f_1}{S_{11}}$$

$$y_i = \frac{1}{S_{ii}} \left(f_i - \sum_{j=1}^{i-1} S_{ji} y_j \right) \qquad i = 2,\ 3,\ \ldots,\ N$$

Algorithmus des Rückwärtseinsetzen:

$$\begin{pmatrix} S_{11} & S_{12} & \cdots & S_{1N} \\ 0 & S_{22} & \cdots & S_{2N} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & S_{NN} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_N \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}$$

$$u_N = \frac{y_N}{S_{NN}}$$

$$u_i = \frac{1}{S_{ii}} \left(y_i - \sum_{j=i+1}^{N} S_{ij} u_j \right) \qquad i = N-1,\ N-2,\ \ldots,\ 1$$

3. Andere Funktionalitäten

Die vorhergegangen Grundfunktionalitäten sind erforderlich, um ein Problem zu lösen. Während der Entwicklung des Programms, haben wir an andere Funktionalitäten gedacht, die auch sehr nützlich sein konnten. Es war unsere Eigeninitiative, sie zu implementieren aber unser Betreuer war immer einverstanden und hat uns beratschlagt. Im Folgenden beschreiben wir die drei Hauptzusätze, die unserer Meinung nach das Programm am meisten bereicherten.

3.1. Knotenumnummerierung

Aufgrund der Wahl der Ansatzfunktionen mit lokalem Träger ist die Steifigkeitsmatrix schwach besetzt. Die Matrixeinträge $K_{ij} = a(p_j,p_i)$ sind nämlich nur von Null verschieden, wenn der Durchschnitt des Inneren der Träger der Ansatzfunktionen p_i und p_j nicht leer ist.

Bei geeigneter Knotennummerierung besitzen die FE-Matrizen eine Bandstruktur. Die Bandweite, d.h. der größte Abstand, den ein Nicht-Null-Element einer Spalte vom entsprechenden Hauptdiagonalelement hat, hängt von der Knotennummerierung ab.

Diese zwei Eigenschaften werden bei der Abspeicherung von K genutzt. Das Programm benutzt die „*variable Bandweite spaltenweise (VBS)*“ oder „*Skyline-Speicherung*“. Dabei werden nur die Elemente abgespeichert, die in der Hülle sind, d.h. zwischen dem ersten Nicht-Null-Element der jeweiligen Spalte und dem Hauptdiagonalelement.

Von der Nummerierung der Knoten hängt es ab, wie groß die Besetztheitsstruktur und damit die Zahl der Elemente, die abgespeichert werden, von der Knotennummerierung ab. Es ist wichtig, dass die Knoten jedes Elements Nummern haben, so dass die Differenz der Knotennummern relativ klein ist. Dies führt zu einer „kleinen“ Länge der Hülle.

Für eine gegebene Knotennummerierung kann das Programm (mit dem Algorithmus von Cuthill-McKee) eine Umnummerierungsstrategie durchführen, um die Länge der Hülle zu reduzieren.

Das Programm kann .net Dateien, die für lineare Ansatzfunktionen geschrieben sind, auch bei quadratischen (oder kubischen, ...) Funktionen benutzen. In diesem Fall sind nur die 3 Eckpunkte

nummeriert und das Programm fügt die zusätzlichen Knoten (3 pro Dreieck für die quadratischen Funktionen) mit den folgenden Nummern hinzu. Es ist aber möglich, die Knoten automatisch umzunummerieren, um den Prozess der Auflösung des Problems schneller zu machen.

Algorithmus von Cuthill McKee:

Die heuristische Begründung des beschriebenen Vorgehens besteht einfach darin, dass benachbarte Knoten möglichst bald im Nummerierungsprozess berücksichtigt werden müssen, andernfalls große Indexdifferenzen auftreten, was zu einer großen Bandbreite führt.
Deshalb beruht beispielsweise die Festsetzung, die Knoten innerhalb einer Stufe fortlaufend unter Berücksichtigung ihres zunehmenden Grades (Unter dem Grad *d(z)* eines Knotens *z* versteht man die Anzahl der benachbarten Knoten) zu nummerieren, auf der einleuchtenden Strategie, dass Knoten mit vielen Nachbarn möglichst hohe Nummern erhalten sollen, um die im nächstfolgenden Schritt auftretenden Indexdifferenzen klein zu halten.

1. Zum Startknoten *r* bestimmt man alle benachbarten Knoten. Diese Knoten werden mit zunehmendem Grad fortlaufend nummeriert. Bei benachbarten Knoten mit gleichem Grad besteht selbstverständlich eine Willkür in der Reihenfolge der Nummerierung. Die in diesem Schritt nummerierten Knoten haben alle die Distanz 1 vom Startknoten *r*. Sie bilden die erste Stufe.

2. Zu den Knoten der ersten Stufe mit aufsteigenden (neuen) Nummern bestimme man sukzessive ihre benachbarten und noch nicht neu nummerierten Knoten und nummeriere sie je mit zunehmendem Grad. Die in diesem Schritt nummerierten Knoten besitzen die Distanz 2 vom Startknoten und bilden die zweite Stufe im Nummerierungsprozess.
   Wiederhole diesen Prozess, bis alle Knoten durchnummeriert sind.

3. Man stellt fest, dass das Profil oft ganz wesentlich verkleinert werden kann, falls die Knotenvariablen exakt in der umgekehrten Reihenfolge durchnummeriert werden, wie sie der oben beschriebene Prozess liefert.

Implementierung des Algorithmus:

(1) Suche den so genannten Startknoten oder Wurzel *r*. Er bekommt den Nummer 1.
$S = \{r\}$

(2) Solange nicht alle Knoten neunummeriert sind,
- $S_{neu} = \{\}$
- Für jeden Knoten *x* von *S*,
  - Zum Knoten *x* bestimme man alle benachbarten und noch nicht neu nummerierten Knoten $\{k_1, k_2, ..., k_r\}$ und nummeriere sie je mit zunehmendem Grad.
  - $S_{neu} = S_{neu} + \{k_1, k_2, ..., k_r\}$
- $S = S_{neu}$

(3) Umkehrung der Nummerierung vermittels der Substitution: $k \rightarrow n+1-k$

Algorithmus zur Bestimmung der Startknote:

Im diesem Schritt zerlegt man die Vernetzung in Stufen oder Schichten.
$R(g) = (L_1, L_2, ..., L_r)$ heißt Stufenstruktur oder Schichtung (engl. level structure) mit der Wurzel *g*, wenn das Folgende gilt:

1. $\bigcup_{i=1}^{r} L_i$ = Menge, welche alle Knoten der Vernetzung enthält
2. L_r ist nichtleer
3. $L_1 = \{g\}$
4. Der kürzer Weg zwischen einem Knoten $h \in L_i$ und *g* besteht aus *i*-1 Kanten

r heißt die Tiefe der Schichtung. Ihre Breite *k* ist die Anzahl der Elemente der Schicht mit den meisten Knoten.

Gibbs, Poole und Stockmeyer haben vorgeschlagen, Stufenstrukturen mit geringer Breite zu benutzen. Sie werden gewöhnlich unter den Stufenstrukturen mit größter Tiefe gesucht. Das heißt, wenn mehr Stufen vorhanden sind, entfallen im Schnitt weniger Knoten auf die einzelnen Stufen.

Es wäre ideal, zuerst einen so genannten Durchmesser des Netzes zu bestimmen, der durch zwei Knoten festgelegt wird, deren kürzester Verbindungsweg im Graphen am längsten ist. Das Problem ist zwar lösbar, meist wird jedoch ein approximativer Algorithmus dafür genutzt.

Mit diesem Algorithmus bildet g nicht mit Sicherheit der Endpunkt eines Durchmessers, doch wird die Strukturtiefe annähernd maximal sein, und man begnügt sich mit einem Pseudodurchmesser.

Dieser erste Schritt ist sehr geeignet, automatisch günstige Startknoten zu bestimmen, da der Aufwand relativ gering ist und in der Regel nur wenige Startpunkte getestet werden müssen, um einen Pseudodurchmesser zu finden.

(1) Wähle einen Knoten g mit minimalem Grad

(2) Erzeuge die Schichtung $R(g) = (L_1, L_2, \ldots, L_r)$. Die Elemente der letzten Schicht L_r seien f_j. Sie seien bez. ihres Grades geordnet, d.h. $d(f_j) \leq d(f_{j+1})$

(3) Setze $j = 1$

(4) Berechne die Tiefe der Schichtung $R(f_j)$: s. Wenn $s > r$ ist, setze man $g = f_j$ und kehre nach (2) zurück

(5) Wenn $j < l$ ist, erhöhe man j um 1 und wiederhole (4)

3.2. Parser

Ein Parser ist ein Programm oder Teil eines Programms (Routine), das Text bzw. Programmcode syntaktisch analysiert und aufgliedert. Dieser Text ist meistens entweder eine Eingabe von einem Nutzer, wie etwa eine Befehlszeile, der Quellcode eines Programms oder eine Datei, die Formatierungsanweisungen (z.B. Tags) enthält.

Das Ziel unseres Parser ist die Werte die Funktion *f* zu finden. Diese Funktion wird als String-Objekt in den Dateien .dat geschrieben (Zum Beispiel *f = 2*X+exp(Y)*) und wir wollen die Werte dieser Funktion in verschiedenen Punkten der Vernetzung kennen, um den Lastvektor zu berechnen.

Zuerst wird das String-Objekt in „Token-Objekte“ zerlegt. Ein „Token“ stellt eine mathematische Funktion (sin(), cos(), usw.), eine Variable (X oder Y), eine Zahl, einen Operator (+, -, usw.) oder eine Klammer dar. Um die Werte der Funktion zu berechnen, braucht das Programm die Koordinaten des Punkts, wo die Funktion berechnet werden soll. Die „Tokens“, die einer Variablen entsprechen, werden durch die Koordinaten ersetzt. Dann wird die Funktion *f* durch den folgenden Algorithmus berechnet:

1. Suche die Klammern, die keine andere Klammern enthalten (die inneren Klammern)
2. Berechne den Ausdruck, der zwischen diesen Klammern steht:
   a. Ersetze die Tokens, die mathematischen Funktionen entsprechen, durch die Werte der Funktion
   b. Rechne alle Multiplikationen und dann alle Additionen
   c. Ersetze den ganzen Ausdruck durch ihre Werte
3. Wenn es noch Klammern gibt, wiederhole 1.
4. Berechne den restlichen Ausdruck (er hat keine Klammern)

Am Ende dieses Prozesses gibt es nur ein Token-Objekt, das einen Wert enthält. Er ist der Wert der Funktion *f*.

3.3. Verfeinerung der Vernetzung

Falls das Programm mit Dreiecken und stückweise linearen Ansatzfunktionen p_i arbeitet, kann auch die Vernetzung verfeinert werden. Nach der Verfeinerung wird eine erneute Problembearbeitung mit der feineren Vernetzung durchgeführt. Wenn eine Verfeinerung gefordert wird, wird eine höchste zugelassene Temperaturabweichung abgefragt. Das Programm verfeinert die Vernetzung, bis alle Temperaturabweichungen zwischen zwei Eckpunkten jedes Dreiecks unter einem gewünschten Grenzwert liegen. Falls „0“ eingegeben wurde, wird die Vernetzung vollständig verfeinert (jedes Dreieck wird in vier kongruente Teildreiecke geteilt) aber nur ein Mal.

Algorithmus der Netzverfeinerung:

(1) Teile alle Dreiecke, für die die Temperaturabweichung zwischen 2 Knoten zu hoch ist, in vier kongruente Teildreiecke, d.h. definiere auf allen Kanten dieser Dreiecke den Kantenmittelpunkt und generiere durch Verbindung dieser Mittelpunkte die Teildreiecke.

(2) Solange mindestens eine Dreiecksviertelungen durchgeführt wurde

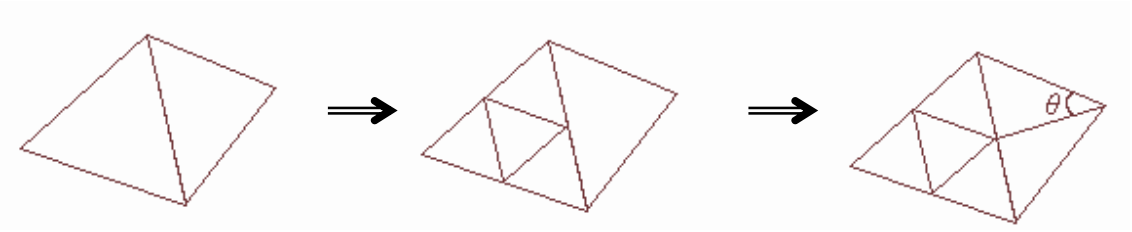
   (i) Teile im zu verfeinernden Netz alle Dreiecke, die nicht geviertelt wurden, bei denen aber auf mindestens zwei Kanten neue Knoten generiert worden sind, in vier kongruente Teildreiecke.

   (ii) Für jedes Dreieck, bei dem genau auf einer Kante ein neuer Knoten generiert wurde,

      - Wenn bei einer Bisektion dieses Dreiecks 2 Dreiecke, die eine schlechte Qualität hätten (d.h. die Innenwinkel θ wäre zu spitz: $\theta<25°$), generiert würden, führe eine Dreiecksviertelung dieses Element durch.

- Sonst teile dieses Dreieck in zwei Teildreiecke, indem dieser Knoten mit dem der entsprechenden Kante gegenüberliegenden Knoten verbunden wird.

(3) Löse das Problem unter Nutzung der neuen Vernetzung. Wenn es noch Temperaturabweichungen über dem gewünschte Grenzwert gibt, gehe zu Schritt (1) zurück



Netzverfeinerung mit Dreiecksviertelung

4. Bilanzen

4.1. Grenzen

Zwar haben wir Methoden genutzt, um die Problemarbeitung zu beschleunigen. Zum Beispiel berücksichtigt die Skyline, dass die Matrix schwach besetzt ist. Es bleibt aber ein Lehrprogramm und kann natürlich nicht mit einem "richtigen" FEM-Programm konkurrieren, was die Geschwindigkeit der Rechnungen betrifft. Diese beiden Programmarten haben nicht das gleiche Ziel und FEJavaDemo ist schnell genug für seine Benutzung.

Dazu war das Programm entwickelt, um ein skalare lineare Differentialgleichungen 2. Ordnung zu lösen, wie zum Beispiel das stationäre Wärmeleitproblem. Das ist genug, um die Methode der finiten Elemente zu verstehen aber das ist natürlich einschränkend. Trotzdem können auch verschiedene physikalische Sachverhalte durch partielle Differentialgleichungen modelliert werden: Probleme aus der Thermodynamik, der Elektrostatik, der Magnetostatik und der Festkörpermechanik, usw. So könnte das Programm verändert werden, um solche Probleme zu behandeln. Das war nicht unsere Priorität. Wir haben gedacht, dass es nicht notwendig war, aber weitere Programmierer werden sich darin hineinknien können.

4.2. Mögliche Weiterentwicklungen

Wir haben die anfänglichen Ziele erreicht und neue Funktionalitäten hinzugefügt, aber das Programm kann natürlich noch verbessert werden. Wir haben an mehrere Möglichkeiten gedacht, aber hatten nicht die Zeit sie zu realisieren.

4.2.1. Weiterentwicklungen in Bezug auf die Vernetzung

Wir haben viel versucht, eine Methode zu implementieren, um die Netzdaten zu generieren. Mit dem Programm soll man nämlich die Koordinaten allen Knoten schreiben und es wäre seht günstig, nur die Lagen der Ränder und die Dichte der Elemente anzugeben. In der Literatur kann

man eine Vielzahl von Arbeiten über Netzgenerierungsalgorithmen finden. Beispiele für automatische Vernetzungsmethoden sind Quadtree-Octree-Methoden, Voronoi-Typ Methoden und Advancing-Front Techniken. In dem Buch unseres Betreuer wird die Grundidee der Advancing-Front-Methode zur Generierung eines Dreiecksnetzes erläutert und es wäre interessant sie zu implementieren, weil man einfach etwas sich vorstellen könnte, um die Bildung der Vernetzung graphisch zu zeigen. Die Elemente werden nämlich vom Gebietsrands ins Gebietsinn gebildet.
Das ist aber nicht so einfach, zu implementieren. Wir haben uns viel über das Problem informiert und haben viele Internet-Seiten und einige Bücher der Bibliothek gelesen, aber wir haben keinen klaren Algorithmus gefunden. Deshalb haben wir vorgezogen, schon das Hauptprogramm fertig zu machen und am Ende hatten wir nicht genug Zeit mehr, um eine solche Funktionalität noch hinzuzufügen. Trotzdem denken wir, dass es sehr interessant wäre, bessere Bücher zu finden, und eine automatische Vernetzungsmethoden mit graphischer Darstellung zu implementieren.
Viele Programme existieren schon, die Netze generieren können und man kann natürlich sie benutzen. Es wäre unsinnig ein Netz eines großen Gebietes mit komplizierter Geometrie manuell zu erzeugen.

Man könnte auch die Wahl der Elemente, die verfeinert werden müssen, verbessern. Um die Geschwindigkeit zu erhöhen und Speicherplatzbedarf zu sparen, ist es nämlich interessant, nicht das ganze Gebiet zu verfeinern, sondern nur wo es nötig ist. Um zu entscheiden, ob ein Element verfeinert werden muss, rechnet zur Zeit das Programm die Temperaturabweichungen zwischen den Eckknoten. Wenn sie zu hoch ist, teilt es das Dreieck in vier kongruente Teildreiecke. Wir hatten diese Idee und unsere Betreuer hatte uns gesagt, dass es interessant war. Das ist aber nicht immer optimal. Man kann sich Dreiecke vorstellen, die dieselbe Temperatur auf den drei Eckknoten haben, obwohl es einen wichtigen Temperatursturz in der Mitte des Dreiecks gibt. In diesem Fall wird das Dreieck nie verfeinert werden. Man könnte a posteriori-Fehlershätzern (zum Beispiel Residuenbasierte Fehlerschätzer) benutzen, um derartiges zu meiden.

4.2.2. Neue FE-Techniken

Wie schon gesagt war es wichtig für unseren Betreuer, dass FEJavaDemo erweiterbar ist, indem neue Arten von Funktionen und Elementen hinzugefügt werden können. Das haben wir immer im Kopf behalten und am Ende hatten wir noch Zeit, um diese Modularitätseigenschaft auszutesten. Wir haben nämlich die quadratischen Ansatzfunktionen und die Vierecke implementiert, obwohl nur die linearen Ansatzfunktionen über Dreiecken verlangt waren. Es hat uns erlaubt, den Code zu verbessern und wir haben jetzt den Eindruck, dass der Zusatz weitere Ansatzfunktionsarten oder Elementsarten ziemlich einfach ist. Von nun an braucht man nicht, den ganzen Quellcode zu beherrschen, um solche Veränderungen zu tun. Dazu haben wir praktische Handbüchern geschrieben, die die verschieden Schritte erklären.

Zur Lösung der FE-Gleichungssystems kann man sowohl direkte als auch iterative Verfahren benutzen. Direkte Auflösungsverfahren sind Algorithmen, die den Lösungsvektor u in endlich vielen Schritten liefern. Iterative Verfahren bestimmen den Vektor u ausgehend von einer Startnäherung u_0 als Grenzwert einer Folge von Näherungslösungen (u_k).Wir wissen, dass die Matrix K symmetrisch und positiv definiert ist und in diesem Fall wird häufig das *Cholesky-Verfahren* eingesetzt, das ein direktes Verfahren ist.
Es wäre auch interessant, wenn die Software eine Methode der konjugierten Gradienten mit Vorkonditionierung, oder ein anderes iteratives Verfahren hätte. So wäre es möglich verschiedene Verfahren zu vergleichen. Jede Methode hat seine eigenen Vor- und Nachteile, und so könnte der Benutzer die Methode, die für sein Problem geeignet ist, finden. Zum Beispiel für ein Problem mit 3593 Unbekannten braucht ein Programm mit einem Cholesky Verfahren (direkte Verfahren) 0.11 Sekunden und 1.31MB Speicherplatz und ein Programm mit einem SOR-Verfahren (iterative Verfahren) braucht 5.97 Sekunden und 193.19kB. Zur Lösung von großdimensionierten Gleichungssystems sind oft iterative Verfahren die effizientesten Lösungsalgorithmen, aber für kleindimensionierte Probleme ist das Cholesky Verfahren eine gute Wahl. Iterative Verfahren sind ein bisschen schwieriger zu implementieren, weil man zum Beispiel entscheiden muss, wieviel Iterationen erforderlich sind, um eine Näherungslösung mit einer vorgegebenen Genauigkeit zu erhalten. Man muss auch sicher sein dass, die Näherungslösungen u_k gegen die Lösung des Gleichungssystems konvergieren.

4.2.3. Anzeige

Wir denken auch, dass die Oberfläche noch verbesserbar ist. Zwar ist sie viel besser als diese des ehemaligen Programms aber man könnte wahrscheinlich noch andere Zeichentricks und Darstellungen hinzufügen. Wir haben beispielsweise gedacht, dass die ganzen Randbedingungen dargestellt werden könnten. Wir meinen, dass Pfeile den Wärmefluss darstellen könnten. Es wäre auch interessant die Temperaturwerte zu sehen, wenn man den Cursor über die Vernetzung bewegt. Es gibt noch viele andere Möglichkeiten, die der nächste Programmierer erdenken können wird.

4.2.4. Vergleich mit der analytischen Lösung

Jetzt kann FEJavaDemo die FE-Lösung einem Wärmeleitproblem finden. Aber um das Programm auszutesten, wäre es auch gut, wenn man eine analytische Lösung die Software geben könnte. Mit dieser Lösung und mit Randbedingungen könnte das Programm die Problemstellungen finden. Dann rechnet die Software die Lösung mit seiner FEM Methode. Es ist interessant die FE-Lösung und die analytische Lösung zu vergleichen, weil es die Unterschiede zwischen Theorie und Praxis zeigt. Es hilft um das Verfahren zu verbessern, aber auch auf pädagogische Gründe um Schwächen der FEM Methode zu betonen. Es zeigt auch den Einfluss von den Parametern (Art der Ansatzfunktionen, Art des Elements, Verfeinerungsniveau der Vernetzung) auf die Genauigkeit der Lösung.

4.3. Aussichten

Dieses Lehrprogramm wird schon in der TU Dresden benutzt und wir denken, dass auch andere Universitäten FEJavaDemo nutzen könnten. Insbesondere wäre es sehr interessant und einträglich, wenn die Lehrenden der Ecole Nationale des Ponts et Chaussées mit der Software arbeiten, um den Unterricht über FEM vorzubereiten. Es würde die Kurse einfach bebildern, und den Studenten helfen, die Methode zu verstehen. Was das erste Semester in der ENPC betrifft, denken wir, dass das Programm beispielsweise in den "Wissenschaftliches Rechnen" und "Mechanik" Kurser genutzt werden könnte. Es wäre auch günstig, wenn es allen Studenten zur Verfügung stehen würde: es wird schon im Internet sein, aber es könnte auch auf jedem Rechner der Schule installiert werden.

Da das Programm "Open-Source" ist, d.h. irgendwer kann die Quelle ändern, um das Programm zu verbessern, hoffen wir auch, dass einige Benutzer Beiträge zu ihm leisten werden. Sonst könnte auch vielleicht nächstes Jahr ein anderer Student der ENPC während eines Praktikums unsere Arbeit weiterführen.

4.4. Persönliche Eindrücke

4.4.1. Jérémie

Dieses wissenschaftliche Praktikum an der Technischen Universität Dresden war für mich eine sehr gute Erfahrung. Ich denke, dass dieses Praktikum ein guter Weg ist, um das erste Jahr zu beenden, weil es die Verbindung zwischen Theorie und Praxis zeigt.
Ich habe meine Mathematik- und Informatikkenntnisse verbessert, aber ich habe auch die Welt der Universität und der Forschung entdeckt. Selbst wenn ich später keine Forschung später machen will, hat diese Erfahrung mir geholfen, um meine Berufsziel zu überlegen. Es war auch persönlich und auf dem Gebiet der Wissenschaft sehr bereichernd.

Zuerst möchte ich noch einmal Professor Walter für seine Hilfe bei der Lösung aller praktischen Probleme danken. Ich will auch das gute Arbeitsklima in dem Institut und die Verfügbarkeit unserer Betreuer betonen. Michael Jung und Kshitij Kulshreshtha haben uns nämlich während des ganzen Praktikums geholfen, was die Mathematik und die Informatik betrifft. Wir hatten auch die Möglichkeit, Unterricht zu besuchen: Deutschkurs mit dem Erasmus Austauschprogramm und Javakurs bei Professor Walter. Dieser Unterricht hat mir erlaubt, meine Kenntnisse zu verbessern und das Leben der Universität besser kennenzulernen.

Am Anfang des Praktikums hatte ich ein großes Interesse für die Programmierung, aber auch für das wissenschaftliche Rechnen und ich wollte ein Praktikum in einem dieser Bereiche absolvieren. Tatsächlich verband dieses Praktikum die beiden Lehrfächer. Also war es für mich ein perfektes Praktikumsthema. Dieses Praktikum war auch interessant, weil es ein vollständiges Projekt ist, das wir vollständig durchgeführt haben. Die Grundidee war natürlich von unserem Betreuer, aber dann haben wir Freiheit gehabt, was die Funktionalitäten und die Struktur des Programms betrifft. Ich habe den Eindruck, dass wir etwas Nützliches gemacht haben, und nicht nur einen Teil eines großen Projekts, das wir nicht völlig verstehen konnten.

Was die Arbeit zu zweit betrifft, war es das erste Mal, dass ich ein so langes Projekt zu zweit machen musste, aber ich denke, dass es sehr gut vorangekommen ist. Am Anfang hatten wir dieselben Informatikkenntnisse und wissenschaftlichen Rechnenkenntisse und es hat uns

ermöglicht, mit derselben Geschwindigkeit zu arbeiten. Zu Beginn haben wir an denselben Algorithmen gearbeitet, somit haben wir beide die Struktur des Programms gut verstanden. Dann haben wir verschiedenen Teilen gemacht, aber immer in Zusammenarbeit.

Ich denke, dass wir mit der Ausbildung des ersten Jahres gute Grundlagen im Bereich des wissenschaftlichen Rechnens und der Informatik gelegt haben. Dennoch war es für uns nötig, Java zu lernen, aber es war nicht zu schwer mit unseren Kenntnissen von C++. Wir sollten auch wissenschaftliches Rechnen weiterstudieren (andere Randbedingungensarten, andere Assemblierungsprozesse usw.). Meiner Meinung nach hat das erste Jahr uns erlaubt, solche Begriffe schnell zu verstehen.
Für die Wahl meiner Fachrichtung („département“) in der ENPC, habe ich zwischen IMI (Informatik, Mathematik) und SEGF (Wirtschaft, Geschäftsführung) lange gezögert. Dieses Praktikum hat mein Interesse für die Programmierung bestätigt, aber es hat mir auch gezeigt, dass ich kein Programmierer werden möchte. Die Programmierung hat während des Praktikums mir sehr gefallen, aber ich meine, dass ich sie nicht als Beruf machen könnte. Zum Schluss halte ich meine Auswahl für SEGF, aber ich hoffe, dass ich nächstes Jahr an ein paar Lehrveranstaltungen der IMI-Fachrichtung teilnehmen werden können.

4.4.2. Aurélien

In jeder Hinsicht war für mich dieses wissenschaftliche Praktikum an der Technischen Universität eine treffliche Erfahrung.

Eine gute Lebenserfahrung, weil es sehr interessant war, drei Monate im Ausland zu bleiben, um die fremden Sitten und Gebräuche zu durchschauen. Ich habe mich bemüht, meine Deutschkenntnisse zu verbessern, indem ich versuchte, viele Leute kennenzulernen, und Deutschunterrichte hörte, der an der Universität gehalten wurde. Unter anderem haben wir auch Dresden, Berlin, Weimar und das Konzentrationslager Buchenwald besichtigt, um von der örtlichen Kultur durchdrungen zu sein.

Was die Arbeit zu zweit betrifft, war dieses Praktikum auch eine sehr bekömmliche Erfahrung, denn ich hatte noch nie soviel mit einer anderen Person an einem Projekt zusammengearbeitet. Meines Erachtens haben wir gemeinschaftlich geschafft, wenn es nutzbringend war, und die Aufgaben halbiert, wenn die Arbeit sich dazu eignete.

Vor allem war dieses Praktikum eine sehr gute Berufserfahrung. Zunächst, weil ich die Möglichkeit nicht ausschließe, später in der Welt der Forschung zu arbeiten und dieses Praktikum konnte mir darüber einen kleinen Einblick geben. Außerdem ist die Entwicklung einer Software von einem Ende zum anderen sehr zufriedenstellend, weil wir das Gefühl haben, etwas, das in der Zukunft genutzt werden wird, getan zu haben. Unser Betreuer hat uns viele Freiheiten gelassen, um eigene Ideen für weitere Funktionalitäten zu implementieren. Was mich betrifft, fand ich, dass es viel anspornender ist, an einem Thema zu arbeiten, auf das man selbst gekommen ist, und dann zu versuchen, den Betreuer zu befriedigen. Hier spreche ich beispielsweise von der Knotenumnummerierung und der Netzverfeinerung.
Ich fürchtete auch, dass dieses Praktikum nur in Programmierung bestand, weil es ein bisschen langweilig sein könnte. In der Tat hatten wir die Gelegenheit, unsere Kenntnisse über die Methode der finiten Elemente zu verbessern. Das war auch eine interessante Fortsetzung unserer Kurse der "1ère année" in Wissenschaftliches Rechnen und in Mechanik. Selbst was die eigentliche Programmierung betrifft, war es oft lehrreich verschiedene Algorithmenmethoden zu lernen, wie die Speicherungstechniken der schwachen besetzten Matrizen. Meiner Meinung nach hat der Umstand, dass wir mit Java und nicht mit C++ programmiert haben, erlaubt eine breitere Sicht zu haben, weil es immer interessant ist, die Struktur verschiedener Programmiersprachen zu vergleichen.

Angesichts meiner Departementswahl, denke ich, dass es schwer wäre, ein geeigneteres Praktikum zu finden. Ich zögere nämlich noch zwischen „IMI“ (Mathematik und Informatik), weil die Begriffe von Modellierung, Simulation und Programmierung mir an sich sehr gefallen, und „GMM“ (Mechanik und Materialen), denn die Mechanik hat mich seit lange begeistert. Deshalb war die Programmierung einer Software, die dank einer Methode, die oft in der Mechanik genutzt wird, um physische Probleme zu behandeln, ideal. Das Praktikum hat meine Meinung gefestigt, dass die Programmierung mich während drei Monaten vergnügt, aber es gibt einen großen Unterschied, wenn es ein Beruf wird.

Literaturverzeichnis

Jung, M.; Langer, U.: *Methode der finiten Elemente für Ingenieure* Eine Einführung in die numerischen Grundlagen und Computersimulation. B.G. Teubner Stuttgart - Leipzig - Wiesbaden 2001

Neundorf, W.; *Wissenschaftliches Rechnen* Matrizen und LGS. Ilmenau 2002
http://www.mathematik.tu-ilmenau.de/~neundorf/education/wr/wr1.pdf

Horstmann, C.; Cornell, G.: *Core Java 2*. Sun Microsystems Press 1999

SunMicrosystems; *Java 2 Platform, API Specification*. Juni 2004
http://java.sun.com/j2se/1.4.2/docs/api/

Technische Universität Dresden. Juni 2004
http://www.tu-dresden.de/

Anlagen

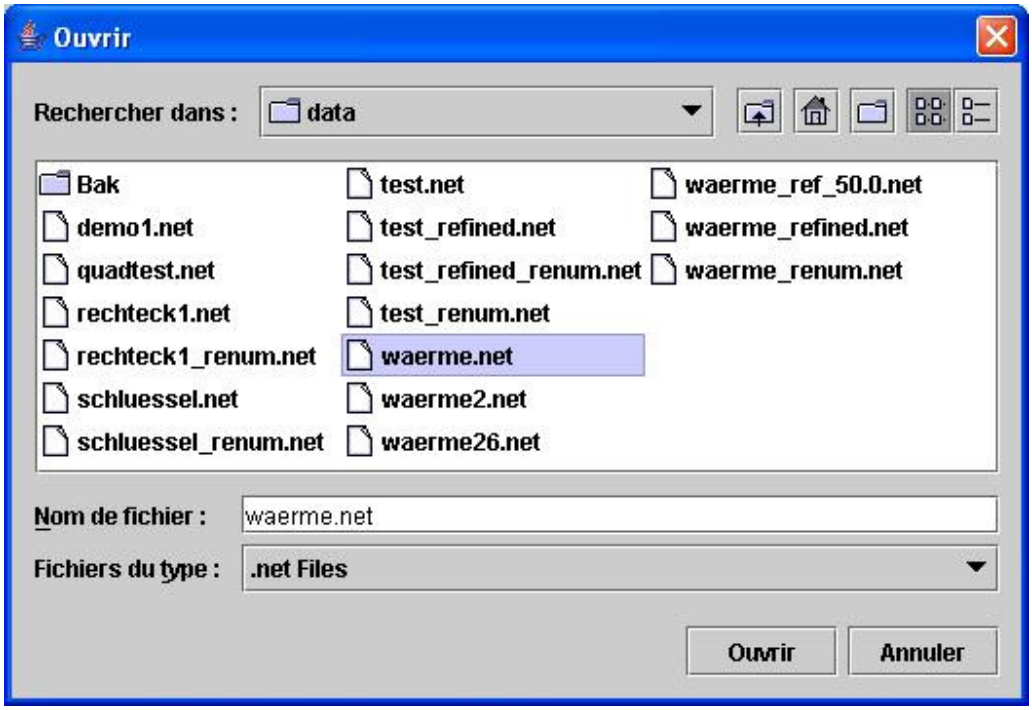
Anlage 1 - Programmablauf

Wahl der Art der Ansatzfunktionen:

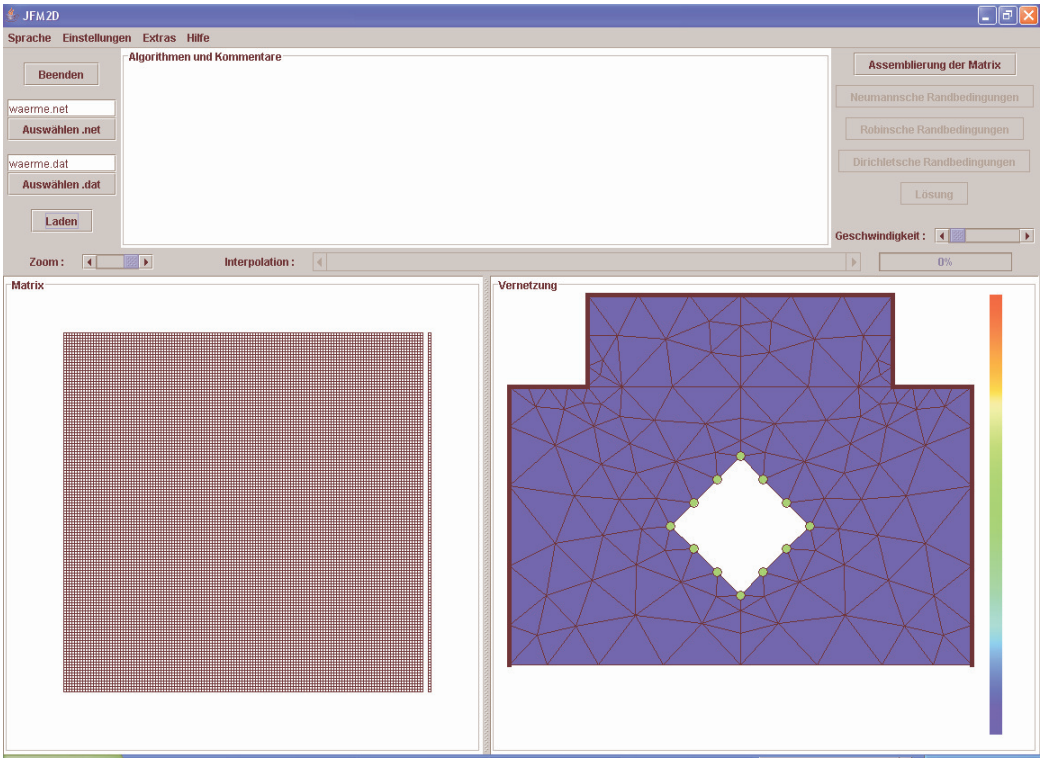
Standardmäßig arbeitet das Programm mit linearen Ansatzfunktionen. Durch den Menüpunkt „Einstellungen → Art der Ansatzfunktionen“ können auch quadratische Ansatzfunktionen gewählt werden.

Einlesen der benötigten Datenfiles:

Durch Anklicken des Knopfs „Auswählen .net“ (bzw. „Auswählen .dat“) werden die zur Verfügung stehenden Netzfiles (bzw. „Dat“-files) angezeigt.



Nach Auswahl den gewünschten Files mittels „Laden“ werden diese geladen.



Aufbau der Steifigkeitsmatrix und des Lastvektors ohne Berücksichtigung der Randbedingungen:

Sobald das Einlesen des Netz- und Datenfiles beendet ist, wird der Knopf „Assemblierung der Matrix“ aktiv. Er dient dazu, die Berechnung und Assemblierung der Steifigkeitsmatrix und des Lastvektors ohne Randbedingungen zu starten.

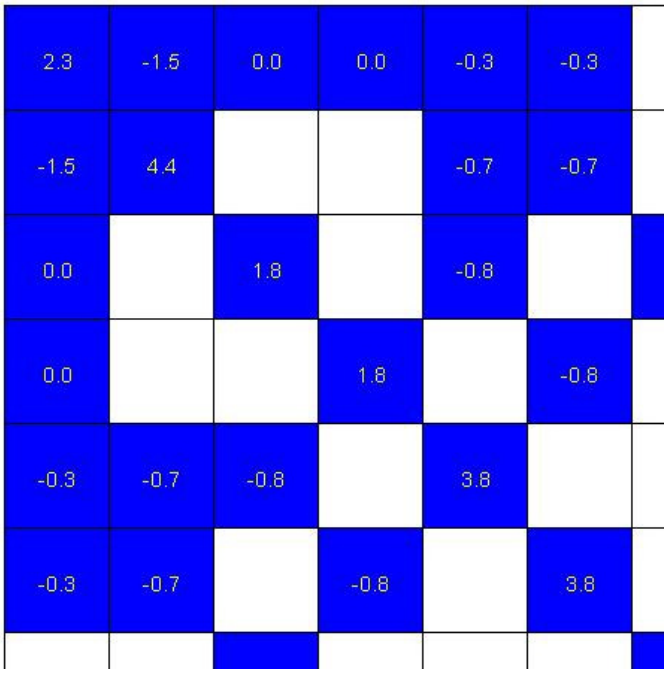
Während dieses Prozesses, erlaubt ein Scrollbar, die Geschwindigkeit der Anzeige zu verändern. Oft zieht man vor, dass das Programm schnell die Lösung liefert. Deshalb werden die Stufen der Rechnung nicht mehr angezeigt, wenn der Rollbalken am weitesten links ist.

Berücksichtigung der Randbedingungen:

Durch Anklicken der drei folgenden Knöpfe werden die Randbedingungen berücksichtigt.

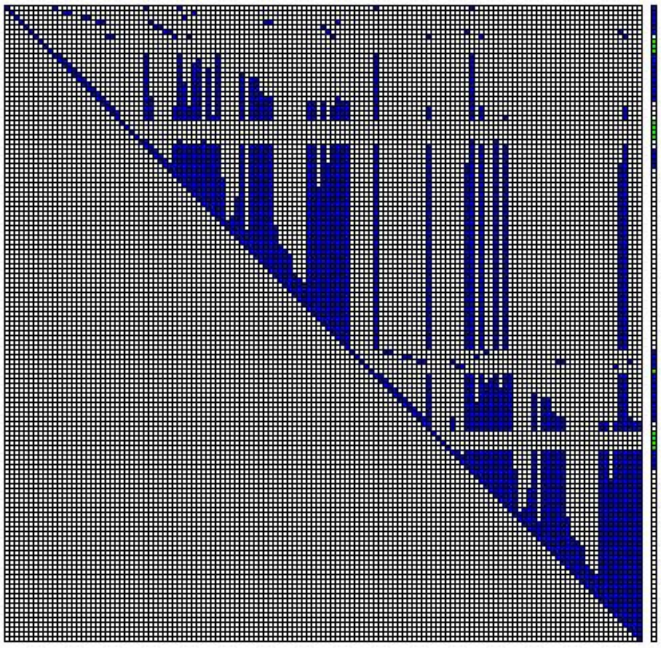
Während der verschiedenen Assemblierungsprozesse, sind die Elemente der Vernetzung und die jeweiligen Matrixeinträge, mit denen das Programm arbeitet, in Rot eingefärbt. Die Nicht-Null-Einträge sind blau, um hervorzuheben, dass die Matrix schwach besetzt ist. Die grünen Elemente entsprechen den Dirichlet-Knoten.

Ein Scrollbar erlaubt, den Zoom auf die Matrix zu verändern. Wenn der Rollbalken am weitesten links ist, werden die Werte der Einträge angezeigt. Zwei Leisten deuten die Nummern der Spalten und Zeilen an.

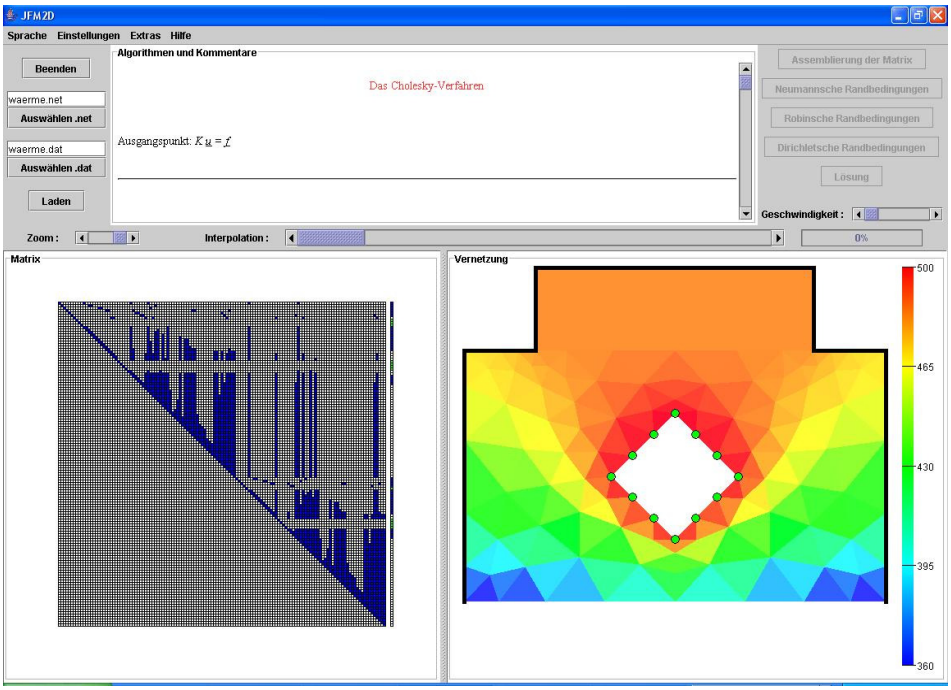


Lösung des FE-Gleichungssystems:

Nachdem alle Randbedingungen berücksichtigt wurden, leitet der Knopf „Lösung“ den Auflösungsprozess ein. Mittels Auswahl des Menüpunktes „Einstellungen → Anzeige → S'S Zerlegung zeigen“, wird zuerst die obere Dreiecksmatrix S der S^TS Zerlegung angezeigt. Ein kleiner Zeichentrick symbolisiert die Durchführung des Vorwärts- und Rückwärtseinsetzens.

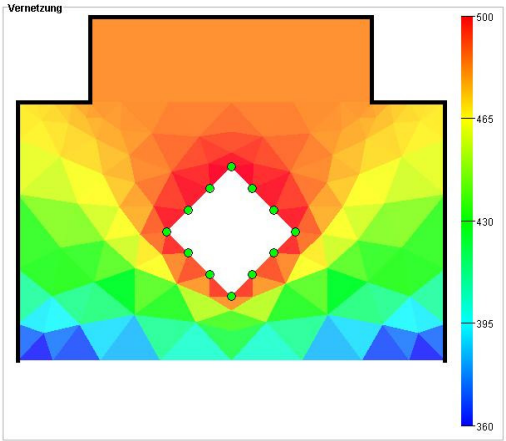


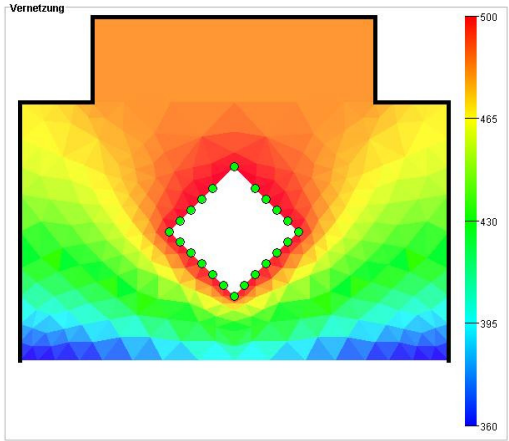
Nachdem die Auflösung beendet ist, werden das Temperaturfeld und eine Temperaturskala dargestellt.



Verfeinerung der Vernetzung:

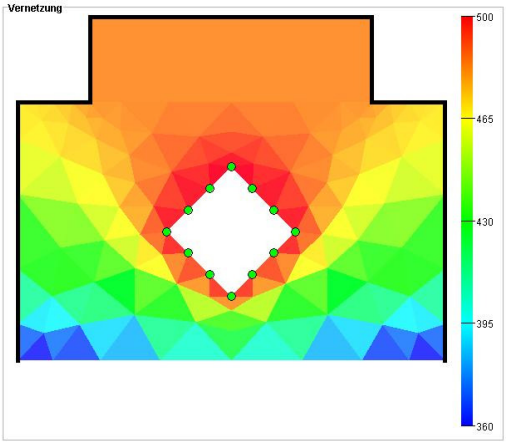
Falls das Programm mit linearen Ansatzfunktionen und Dreiecken arbeitet, kann auch die Vernetzung verfeinert werden („Extras → Verfeinerung der Vernetzung“). Wenn eine Verfeinerung gefordert wird, wird eine höchste zugelassene Temperaturabweichung zwischen zwei Eckpunkte abgefragt. Falls „0“ eingegeben wurde, wird die Vernetzung vollständig verfeinert aber nur ein Mal. Es gibt die Möglichkeit, die Dateien der neuen Vernetzung zu erstellen („Extras → Neue Dateien erstellen ... → ... für Verfeinerung“).

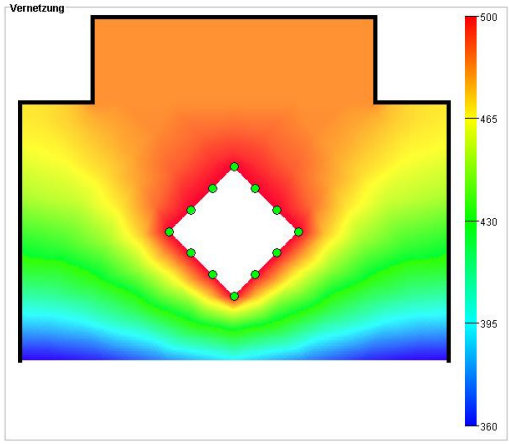




Interpolation der Lösung:

Durch den Scrollbar „Interpolation“ wird die Lösungsdarstellung verbessert: die Elemente können intern wiederholt zerlegt werden und die Lösung wird über den neuen Knoten linear interpoliert. Da es ziemlich lang dauern kann, gibt es einen Fortschrittsbalken, der dazu dient, ein Feedback über den aktuellen Status der Arbeit zu haben.



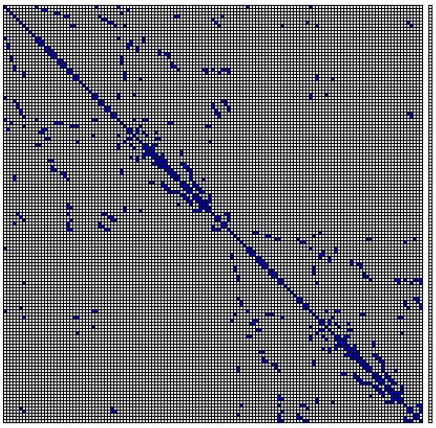


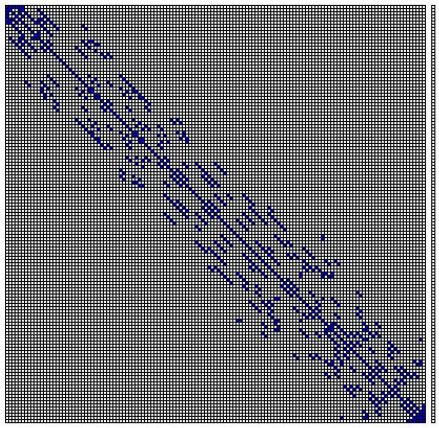
Speicherung der Lösung:

Die Lösung kann (d.h. die Knoten mit Ihren Koordinaten und dem Temperaturwert) mittels „Extras → Lösung speichern“ Menüeintrag gespeichert werden.

Umnummerierung der Knoten:

Die Knoten können auch durch das „Extras → Knotenumnummerierung“ Menü umnummeriert werden. Der Menüeintrag „Automatische Knotenumnummerierung bei quadratischen Funktionen“ ist standardmäßig ausgewählt. Er erlaubt den Prozess der Auflösung des Problems schneller zu machen. Es gibt die Möglichkeit, die Dateien der neuen Vernetzung zu erstellen („Extras → Neue Dateien erstellen ... → ... für Umnummerierung“).



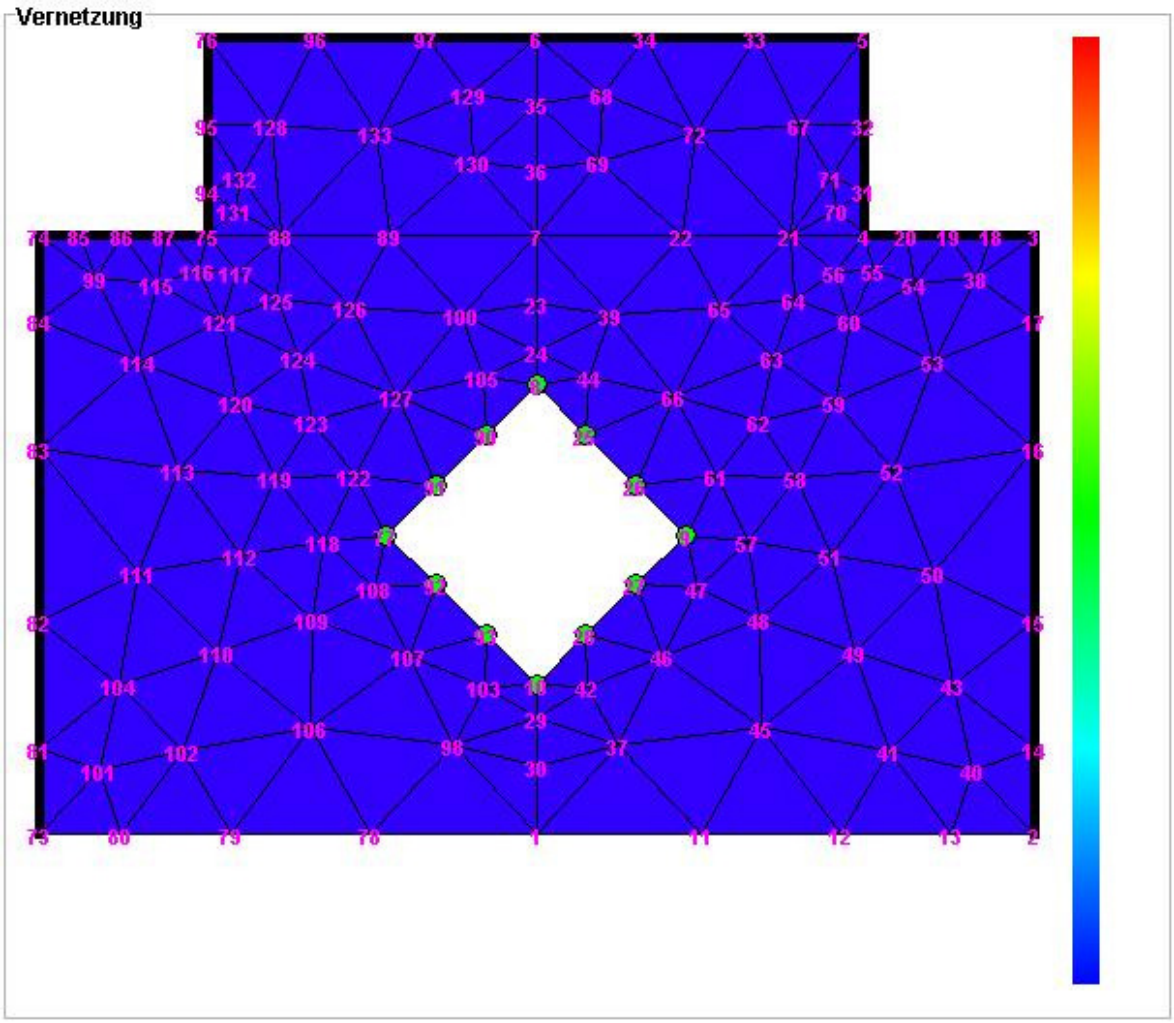


Andere Möglichkeiten:

Natürlich kann die Software dank dem „Sprache“ Menü übersetzt werden.

Zwischen jedem Schritt können Jpeg-Files erstellt werden, um Bilder der Matrix und der Vernetzung zu erhalten („Extras → Jpeg-Files erstellen“).

Dank dem „Einstellungen → Anzeige“ Menü, ist es durch den „Randbedingungen zeigen“ Menüeintrag möglich, die Dirichlet-Knoten in grün einzufärben und die wärmeisolierten Kanten zu betonen. Dazu können die Knotennummern angezeigt werden („Einstellungen → Anzeige → Knotennummern anzeigen“).



Anlage 2 – Struktur der Eingabefiles

Um eine Vernetzung bereitzustellen, muss man Datenfiles schreiben: ein „.net“ File und ein „.dat“ File. Das erste File enthält Informationen über die Vernetzung und der zweite über die Randbedingungen, die Wärmeleitzahlen und die Funktion *f*.

Das .net File

Linien, die mit # beginnen, sind Kommentar und das Programm liest sie nicht.

In der ersten Zeile wird die Art der Elemente angegeben (1 für Dreiecke, 2 für Vierecke usw.).

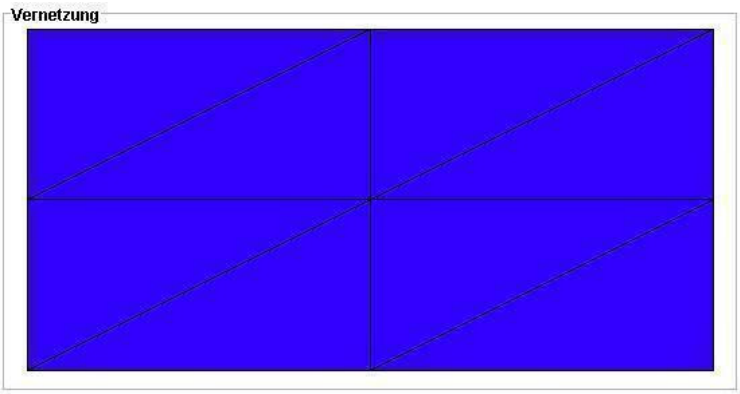
Dann gibt es die Anzahl die Knoten (d.h. die Dimension der globalen Steifigkeitsmatrix) und die Anzahl der Elemente.

Für die Definition der Knoten müssen drei Zahlen angegeben werden: die globale Nummer, die X-Koordinate und die Y-Koordinate.

Dann werden die Elemente mit ihrer globalen Nummer, den globalen Nummern ihrer Eckpunkte (drei Nummern für Dreiecke, vier für Vierecke, ...) und die Materialbereichsnummer definiert.

Die nächste Zeile enthält die Anzahl der Randkanten und dann folgen die Definitionen der Randkanten. Sie werden mit ihrer globalen Nummer und den zugehörigen Anfangs- und Endknoten definiert.

Das folgende Beispiel ist das .net File einer einfachen Vernetzung



```
# FEJavaDemo - Netzfile
#
# Art der Elemente (1 für Dreiecke, 2 für Vierecke usw.)
  1
#
# Anzahl der Knoten und Anzahl der Elemente
  9 8
#
# Globale Nummer und Knotenkoordinaten
  1  0.0  0.0
  2  0.0  1.25
  3  0.0  2.5
  4  2.5  0.0
  5  2.5  1.25
  6  2.5  2.5
  7  5.0  0.0
  8  5.0  1.25
  9  5.0  2.5
#
# Globale Nummern der Knoten jeder Elemente (und Nummer des MaterialBereich)
  1  4  5  1  1
  2  2  1  5  1
  3  5  6  2  1
  4  3  2  6  1
  5  7  8  4  1
  6  5  4  8  1
  7  8  9  5  1
  8  6  5  9  1
#
# Anzahl der Randkanten
  8
#
# Anfangs- und Endknoten der Randkanten (und globale Nummer)
  1   1  4
  2   4  7
  3   7  8
  4   8  9
  5   9  6
  6   6  3
  7   3  2
  8   2  1
```

Das .dat File

Am Beginn des Files werden die Anzahl der Materialbereiche und dann die Wärmeleitzahlen λ_1 und λ_2 in jedem Bereich geschrieben.

Dann werden die Randbereiche definiert:

Zuerst wird die Anzahl der verschiedenen Randbereiche geschrieben.

Für jeden Bereich sind zwei Zahlen einzugeben: die Anzahl der Kanten und die Art der Randbedingung (1 für Dirichlet, 2 für Neumann und 3 für Robin).

Nun kommen die globalen Nummer der Kante jeder Randbereich und die zugehörigen Randdaten.

- Für die Dirichletschen Randbedingungen muss man zwei Werte angeben, die die Temperatur im Anfangs- und Endknoten sind.
- Was die Neumannschen Randbedingungen betrifft, wird nur ein Wert eingegeben, der der Wärmefluß auf der Kante darstellt $\left(\frac{\partial u}{\partial N} = a \right)$
- Und für die Robinschen Randbedingungen muss man zwei Werte angeben: *a* und *b* $\left(\frac{\partial u}{\partial N} = a \left[b - u(x) \right] \right)$

Die letzte Information dieser Datei ist die Funktion *f*. Sie kann verschieden für jeden Bereich sein und wird als Funktionsausdruck geschrieben (z.B. 2*x+tan(y)+7). Die Funktionen, die genutzt werden können, sind die folgende: tan(), sin(), cos(), exp(), ln(), sqrt() und die Konstante π.

Das folgende Beispiel ist das .dat File, das dem ersten Beispiel entspricht:

```
# FEDemoJava - .dat Datei
#
# Anzahl der Materialbereiche
  2
# Lambda1 und Lambda 2 in MaterialBereich 1
  200.0  200.0
# Lambda1 und Lambda 2 in MaterialBereich 2
  100.0  100.0
#
# Anzahl der verschiedenen Randbereiche
  3
# Anzahl der Kanten und Art der Randbedingung in Randbereich 1
  4  1
# Anzahl der Kanten und Art der Randbedingung in Randbereich 2
  2  2
# Anzahl der Kanten und Art der Randbedingung in Randbereich 3
  2  3
#
# Globale Nummer und Randdaten (a (und b) Koeff) der Kante im Randbereich 1
  1   0.   10.
  2  10.   20.
  3  20.   30.
  4  30.   40.
# Globale Nummer und Randdaten (a (und b) Koeff) der Kante im Randbereich 2
  5   5.
  6   7.
# Globale Nummer und Randdaten (a (und b) Koeff) der Kante im Randbereich 3
  7  10.   50.
  8  10.   50.
#
# Funktion F(x,y) in MaterialBereich 1
  3*cos(x*y*Pi)
# Funktion F(x,y) in MaterialBereich 2
  0
```