Prof. Dr. Karsten Urban
Iris Häcker

SoSe 2013
Abschlussprojekt
ab 24.06.2013

Wima Praktikum 1

Abschlussprojekt Gruppen 1a und 2a bei Thomas Emberger

Aufgabe 3

Voraussetzungen zum Bestehen des Projektes:

1. Bringen Sie Ihren Studentenausweis mit!
2. Berechnen Sie $A = \mod(M, 3) + 1$, wobei M für **Ihre** Matrikelnummer steht. Sie **müssen** als Abschlussprojekt dann die Aufgabe A bearbeiten! Achtung: Wenn Sie eine andere Aufgabe bearbeiten, so wird dies nicht gewertet! Achten Sie bitte darauf, dass Sie das Abschlussprojekt der richtigen Gruppe nehmen!
3. Sie müssen die Aufgaben selbstständig bearbeiten. Kopierte Lösungen führen zum Nichtbestehen **aller** beteiligten Personen!
4. Ihr Abschlussprojekt muss vollständig funktionieren (Anforderungen siehe in der Projektbeschreibung).
5. Ihr Abschlussprojekt muss schneller sein, als die in der Projektbeschreibung angegebene Schranke. Falls Sie das nicht beim ersten Versuch schaffen, nutzen Sie die Funktionen `tic` und `toc`, um langsame Programmteile zu identifizieren und beseitigen Sie diese anschließend. Alternativ dazu können Sie sich auch mit dem Matlab-Profiler vertraut machen.
6. Jede Funktion die Sie schreiben, muss ausreichend dokumentiert sein, d.h bei Aufruf von `help` <funktionsname> sollten **mindestens** die folgenden Informationen verfügbar sein:
   1. Was macht die Funktion?
   2. Was sind die Eingabeparameter und wie sind sie zu benutzen?
   3. Was sind die Ausgabeparameter und wie sind sie zu benutzen?
7. Die Benutzung von Matlab-Toolboxen ist nicht erlaubt.
8. Bis spätestens **20 Uhr** des letzten Praktikums müssen Sie (einzeln) die Tutoren davon überzeugen, dass Ihr Programm die Anforderungen erfüllt und dass Sie mindestens folgende Bereiche gut beherrschen:
   1. Rechnen mit Vektoren und Matrizen.
   2. Logische Vektoren und Matrizen.
   3. Programmierung (`for / while / if / switch ...`).
   4. Funktionen.
   5. Debugging.
   6. Laufzeitoptimierung.
   7. Plotten und Animationen.
   8. Grundlegendes Matlab-Verständnis (wo werden Variablen abgelegt, wie löscht man Speicher, etc...).

9. Falls Sie inhaltliche Fragen haben oder Hilfe benötigen, können Sie sich gerne an die Tutoren oder die Übungsleiterin wenden. Für Fragen können Sie gerne auch während der anderen Praktikumszeiten vorbeischauen.

   Tutoren

   - Thomas Emberger (thomas.emberger@uni-ulm.de)
   - Stefan Haag (stefan.haag@uni-ulm.de)
   - Clemens Kraus (clemens.kraus@uni-ulm.de)
   - Bayram Memis (bayram.memis@uni-ulm.de)
   - Katharina Schachmatov (katharina.schachmatov@uni-ulm.de)
   - Stefan Schelling (stefan.schelling@uni-ulm.de)

   Wir werden insbesondere inhaltliche Fragen zu den Projekten beantworten sowie Hilfestellung geben wenn dies erforderlich ist. Fragen wie zum Beispiel: Wie geht das? Was macht diese Funktion etc. werden nicht beantwortet! Bevor Sie also Fragen stellen,

   - nutzen Sie das Skript!
   - nutzen Sie die Matlab Hilfe!
   - testen Sie anhand von Minimalbeispielen was einzelne Funktionen machen!
   - Oft finden sich auch schnell gute Antworten auf Fragen bei google!

   *T*ipp: Fangen Sie rechtzeitig (= möglichst bald) mit der Bearbeitung des Projektes an!

Projektaufgabe — Sudoku

Sudoku ist ein Logikrätsel und ähnelt lateinischen Quadraten. In der üblichen Version ist es das Ziel, ein 9×9-Gitter mit den Ziffern 1 bis 9 so zu füllen, dass jede Ziffer in jeder Spalte, in jeder Zeile und in jedem Block (3×3-Unterquadrat) genau einmal vorkommt. Ausgangspunkt ist ein Gitter, in dem bereits mehrere Ziffern vorgegeben sind.

	3							
			1	9	5			
	9	8					6	
8				6				
4					3			1
				2				
	6					2	8	
			4	1	9			
							7	

Abbildung 1: Beispiel eines 9×9 Sudokus

Regeln und Begriffe
Das Spiel besteht aus einem Gitterfeld mit 3×3 Blöcken, die jeweils in 3×3 Felder unterteilt sind, insgesamt also 81 Felder in 9 Zeilen und 9 Spalten. In einige dieser Felder sind schon zu Beginn Ziffern zwischen 1 und 9 eingetragen ("Lösungszahlen"). Typischerweise sind 22 - 36 Felder von 81 möglichen Feldern vorgegeben. Ziel des Spiels ist es nun, die leeren Felder des Rätsels so zu vervollständigen, dass in jeder der je neun Zeilen, Spalten und Blöcke jede Ziffer von 1 bis 9 genau einmal auftritt. Die drei Bereiche (Zeile, Spalte, Block) werden zusammengefasst als *Einheiten* bezeichnet. Solange das Sudoku nicht gelöst ist, können innerhalb einer Einheit mehrere Möglichkeiten für verschiedene Ziffern bestehen. Werden diese Möglichkeiten notiert, nennt man diese *Kandidaten*.

Varianten
Neben dem bekannten9×9 Sudoku existieren noch zahlreiche andere Varianten, bspw. Fudschijamas (Sudokus mit 4×4-Blöcken, also 256 ($=16 \times 16$) Feldern in die je 16 verschiedene Zahlen verteilt werden) und Mini-Sudokus (mit 2×2 Blöcken, also 16 ($=4 \times 4$) Feldern) für Einsteiger.

Lösungsmethoden
Neben Analytisch-systematischen Basismethoden (Scannen, Auszählen, Eliminierung, Kombination) eignen sich zur Programmierung vor allem Mathematische Methoden wie bspw. das *Backtracking*

Backtracking
Auf dem Computer kann man ein Sudoku mit der Backtracking-Methode lösen. Beginnend mit dem ersten freien Feld, probiert man systematisch, mit der Eins beginnend, ob man zu einer Lösung kommt. Beim ersten Widerspruch geht man zurück (engl. backtrack). Dieser Lösungsweg lässt sich sehr elegant *rekursiv* formulieren, und man ist sicher, dass alle Kombinationsmöglichkeiten abgesucht werden. Da es sich um tausende Wege handeln kann, ist dieser Algorithmus nur für Computerprogramme geeignet. Der Lösungsalgorithmus ist allerdings bestimmt nicht der Schnellste, da er keinerlei analytische Vorinformationen verwendet und nur durch Ausprobieren vorgeht. Dennoch erhält man auf gewöhnlichen PCs auch für schwierige 9×9-Sudokus die Lösung innerhalb einer Sekunde. Bei größeren Sudokus stößt die Backtracking-Methode jedoch schnell an ihre Grenzen. Modifiziert man diese Methode dahingehend, dass man nicht versucht, das erste freie Feld zu belegen, sondern ein

Feld mit der kleinsten Anzahl von Kandidaten, dann reduziert sich der Aufwand in der Praxis auf ungefähr lineare Laufzeit, da in der Praxis (auch bei schweren Sudokus) fast immer ein Feld existiert, für das nur eine Zahl in Frage kommt.

Quellen
Obigen Text und Ausführungen über die vorgestellten Varianten und Lösungsverfahren finden Sie auf folgender Seite:

`http://de.wikipedia.org/wiki/Sudoku`

Aufgabe

1. Schreiben Sie ein Matlab-Programm, welches in der Lage ist, aus Text-Dateien Sudokus in Form von Matrizen einzulesen und diese korrekt zu lösen. Implementieren Sie zunächst eine Funktion `function C = Kand(A)`, welche als Input-Operator ein unvollständiges Sudoku als 9×9-Matrix bekommt. Output-Operator soll ein `Cell-Array` C sein, in dem alle möglichen *Kandidaten* der noch nicht besetzten Felder gespeichert werden. Weiter sollen Sie eine Funktion `function A = SudSolv(A)` implementieren, welche als Input-Operator ein unvollständiges Sudoku als 9×9-Matrix bekommt und als Output-Operator dieses vollständig gelöst ausgibt. Algorithmus der `function A = SudSolv(A)` :

   - Bestimmen Sie die möglichen Kandidaten aller leeren Felder des Sudokus
   - Füllen Sie alle leeren Felder des Sudokus aus, die nur einen möglichen Kandidaten haben
   - Kontrollieren Sie, ob die so eingetragenen Werte mit den Sudokuregeln konform sind
   - Iterieren Sie die ersten 3 Schritte bis nur noch Felder mit mehreren möglichen Kandidaten existieren
   - Setzen Sie nacheinander alle möglichen Kandidaten des ersten leeren Feldes ein und beginnen Sie den Algorithmus von vorne bis eine Lösung gefunden wird (Backtracking-Methode)
   - Geben Sie die Lösung aus

   Schreiben Sie eine `main`, in der Sie die beigefügten Sudoku-Text-Dateien

   Sudoku1.txt,Sudoku2.txt,Sudoku3.txt

   einlesen und mit ihnen die Funktionen testen.

2. Erweitern Sie die beiden Funktionen, sodass nun auch Sudokus beliebiger Größe gelöst werden können. Voraussetzung für die Sudokus ist eine mögliche Untergliederung in quadratische Teilmatrizen (d.h. mögliche Sudokus sind: 4×4-Sudokus (4 2×2-Blöcke), 9×9-Sudokus (9 3×3-Blöcke), 16×16-Sudokus (16 4×4-Blöcke),...). Schreiben Sie dazu eine Funktion

   `function A = erwSudSolv(A)`,

   welche die Funktionen

   `function C = Kand(A)` und `function A = SudSolv(A)`

   vereint. Als Input-Operator soll sie das unvollständige Sudoku, als Output-Operator das gelöste Sudoku ausgeben. Achten Sie darauf, dass bei falscher Eingabe (z.B. einer 8×8-Matrix) entsprechend die Funktion per Fehlerbenachrichtigung unterbrochen wird. Erweitern Sie ihre `main`, so dass auch die restlichen Sudoku-Text-Dateien

   MiniSudoku.txt,Sudoku.txt,Fudschijama.txt,unvollstSudoku.txt

   eingelesen und gelöst werden, bzw. bei falscher Eingabe eine Fehlernachricht ausgegeben wird. Die Laufzeit für das Lösen des Fudschijamas sollte kleiner als 13 Sekunden sein.

3. In dieser Teilaufgabe soll ein Sudoku-Generator implementiert werden, der selbst 9×9-Sudokus generieren kann. Schreiben Sie ein `Skript` namens `SudGen`, welches zufällige, vollständig gelöste Sudokus erstellt, diese nach und nach durch Entfernung der Elemente reduziert, bis sie nur noch wenige bereits gelöste Felder (≤ 20) enthalten. Achten Sie darauf, dass Sudokus **eindeutig** gelöst werden müssen. Diese Eindeutigkeit darf bei der Reduzierung nicht verloren gehen.

   Tipp: Zum Erstellen von zufälligen, vollständig gelösten Sudokus sollte ihr `Skript` zuerst Zufallszahlen (zwischen 0-9) generieren (`randi`) und diese zufällig in eine leere 9×9-Matrix einlesen. Mit Hilfe der Funktion `function A = SudSolv(A)` kann man diese dann vervollständigen.