Prüfungsprotokoll Mathematik

Fach: Scientific Computing ☐ Bachelor
Studiengang: Scientific Computing ☒ Master

(Sonstiges bitte von Hand eintragen.)

Prüfer/in: Jürgen FUHRMANN Beisitzer/in: Olivier Sète
Datum: 5.3.2015 Note: 1,0
Prüfungsdauer: 28 min Anzahl der Kandidaten: 1

Vorbereitungszeit: 2 Wochen
Literatur: Vorlesungsfolien, Skripte „Numerik II für Ingenieure", „Numerische Lineare Algebra I", „Nichtlineare Optimierung"

Beurteilung der Prüfung und des/r Prüfers/in:

Herr Fuhrmann lässt einen viel reden, wobei ich irgendwann mündlich wurde, ob ich immer noch weiterreden soll. Aber ich denke, dass er es gut findet, umso mehr man von selbst erzählt. Außerdem hatte ich das Gefühl, dass es sich super spontan überlegt, was er gerade wissen will, ob man es weiß.

Fragen:

Was kann man machen, wenn man das Heat Problem mit Robin boundary conditions hat? → FEM, FVM

Und wie sieht das Problem aus? Wie macht man FVM?

→ $-\nabla \cdot \kappa \nabla u = f$ in Ω

$\kappa \nabla u \cdot n + \alpha(u-g) = 0$ on $\partial\Omega$

Die Idee ist nun, dass wir Ω in control volumes unterteilen können & für jedes c.v gilt der Satz von Gauß. Zusammen mit den boundary conditions kommen wir auf eine Matrix-Gleichung, die es zu lösen gilt. Hab die Formeln aufgeschrieben & immer ein bisschen gesagt, was ich grade umforme:

$$0 \underset{\text{Gauß}}{=} \int_{\omega_k} (-\nabla \cdot \kappa \nabla u - f)\, dx =$$

$$= -\int_{\partial\omega_k} \kappa \nabla u \cdot n_k \, dy - \int_{\omega_k} f \, dx$$

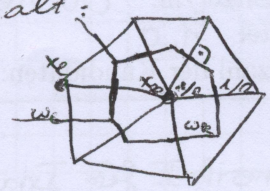
$$= -\sum_{e \in \mathcal{N}_k} \int_{\sigma_{ke}} \kappa \nabla u \cdot n_{ke} \, dy - \int_{\gamma_k} \kappa \nabla u \cdot n \, dy - \int_{\omega_k} f \, dx$$

$$\approx \underbrace{\sum_{e \in \mathcal{N}_k} \frac{|\sigma_{ke}|}{h_{ke}} (u_k - u_e) + |\gamma_k| \alpha (u_k - g_k)}_{} - |\omega_k| f_k$$

→ A: warum kriegen wir hier eine M-matrix?
- off-diagonal entries non-positive
- diagonal entries positive
- idd wenn wir $|\gamma_k| \alpha(u_k)$ dazunehmen.

Wie kommt man an die control volumes?

→ Hier hab ich ein bisschen rumgestottert, dass wir die Voronoi-Zellen einer Delaunay-Triangulierung nehmen kann & man praktischerweise nur die Delaunay-Dreiecke zu speichern hat. Unsere collocation points sind die Ecken der Dreiecke, hab ein bisschen gemalt:



Was macht die Triangulierung Delaunay?

→ Hier hab ich etwas von Winkel $\leq 90°$ gesagt, was natürlich ~~nur~~ die weak acuteness ist. Daraufhin hab ich Kreise um Punkte gemalt, allerdings nur um $\mathcal{C}$: • $\mathcal{D}$, worauf- hin er meint, das stimme noch nicht ganz, also: ⊘.

Wenn wir dann die Matrizen haben, wie kommen wir zur Lösung?

→ Direkte Löser, Iterative Löser oder z.B. CG.
Direkte Löser haben Probleme der Laufzeit & Speicherbedarf.
Die klassischen iterativen Methoden konvergieren relativ langsam, also CG.

Wie erkennt man die Konvergenzrate der iterativen Methoden?

→ $u_{k+1} = u_k - M^{-1}(Au_k - b)$, also mit $\hat{u}$ exakte Lsg

$$u_{k+1} - \hat{u} = u_k - \hat{u} - M^{-1}(Au_k - A\hat{u}) = (I - M^{-1}A)(u_k - \hat{u}) =$$
$$= \ldots = (I - M^{-1}A)^k (u_0 - \hat{u})$$

⇒ Konvergenzrate: $\varrho(I - M^{-1}x)$ muss < 1 sein!

Was ist die Idee von CG?

→ Gradientenverfahren: Richtung orthogonal, aber wenn man $Ax = b$ lösen möchte, entspricht das dem Minimierungsproblem $\min f(x) = \frac{1}{2} x^T A x - b^T x$ und die sind in der A-Norm (A spd.!) die Niveaulinien von f Kreise um die Lösung
⇒ A-orthogonale Richtungen
Da quadratische Minimierung, bekommen wir die exakte Schrittweite und sehen, dass wir die neue Suchrichtung als Linearkombination von d_{k-1} und r_k bekommen, brauchen auch nicht alle die ich speichern!

Bei CG hab ich so viel wie möglich geredet, um möglichst wenig aufschreiben zu müssen.

Wie ist nun die Konvergenzrate von CG verglichen mit dem Gradientenverfahren?

→ Für CG bekommen wir

$$\|e_i\| \leq 2 \cdot \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^i \|e_0\|$$

wobei die Wurzel eben den großen Unterschied macht.

Dann meinte Oliver, dass wir von der Zeit her theoretisch aufhören können. Da hat Herr Fuhrmann kurz überlegt & gesagt, er würde gerne noch was zu Parallelisierung fragen will.

→ Ich hab also auch hier einfach erzählt, was mir in den Kopf kam: Es gibt verschiedene Strukturen, wie wir das Memory verteilen: Shared, Non-Uniform Memory Access, Distributed, Distributed mit Software-Layer. Abhängig davon müssen wir uns Gedanken über Memory-Access machen, zB für Distributed sehr explizit. Es kann zu write-conflicts kommen → mutexes. Problem, dass es dann eventuell wieder sequentiell verarbeitet wird & Dead-Locks

⇒ Reduktion mit Reduktionsvariable, Zwischenergebnisse werden gespeichert & in der main kombiniert.