



### Programmierblatt 3.

Bearbeiten bis: **Sonntag, 24.11.2019**

Während auf dem zweiten Blatt das Poisson-Problem

$$\begin{aligned} -\Delta u(\mathbf{x}) &= f(\mathbf{x}) && \text{für } \mathbf{x} \in \Omega, \\ u(\mathbf{x}) &= 0 && \text{für } \mathbf{x} \in \Gamma_D, \\ \langle \nabla u(\mathbf{x}), \mathbf{n}(\mathbf{x}) \rangle &= 0 && \text{für } \mathbf{x} \in \Gamma_N, \end{aligned}$$

auf dem durch ein Dreiecksnetz  $\mathcal{T}$  gegebenen Gebiet  $\Omega$  schon numerisch gelöst wurde, haben wir dort zum Lösen des resultierenden Gleichungssystems das CG-Verfahren angewendet.

In dieser Programmieraufgabe wollen wir nun das Mehrgitterverfahren anwenden. Hierzu werden wir zuerst eine erweiterte Dreiecksnetzbeschreibung einführen, welche die ganze Hierarchie von Netzen speichert die man ausgehend von einem groben Grundnetz durch mehrmaliges Verfeinern erhält. Insbesondere werden dabei auch die Prolongationsmatrizen gebildet, die den Koeffizientenvektor einer Finite-Element-Funktion von einem Netz auf ein einmal feineres Netz abbilden. Danach erfolgt dann die Implementierung des sogenannten V-Zyklus. Viele der bereits implementierten Funktionen können jeweils als Startpunkt für die neu zu implementierenden Funktionen benutzt werden.

### Mehrgitter-Dreiecksnetze

Ein Mehrgitter-Dreiecksnetz kann kanonisch durch eine *Punktliste*  $\mathbf{P}$  und eine *Liste von Elementlisten*  $\{\mathbf{F}_j\}_{j=1}^J$  dargestellt werden. Bei einer Verfeinerung werden wie gewohnt die neuen Eckpunkte einfach hinten an die Punktliste angehängt; die neue Elementliste, die durch Verfeinerung der vorher feinsten Elementliste gewonnen wird, wird ein neues Element der Liste von Elementlisten. Wie schon bei dem ersten Blatt ist zur Beschreibung der Randbedingungen auch ein Zeilenvektor  $\mathbf{B}$  zu führen, der den Typ der Eckpunkte angibt.

Zusätzlich wird weiter eine *Liste von Prolongationsmatrizen*  $\{\mathbf{T}_j\}_{j=1}^{J-1}$  gespeichert. Die Prolongationsmatrix beschreibt dabei jeweils die lineare Abbildung, welche dem Koeffizientenvektor einer Finite-Elemente Funktion den Koeffizientenvektor zuordnet, der die gleiche Funktion auf dem einmal verfeinerten Netz darstellt.

Ein solches Mehrgitter-Dreiecksnetz, welches eine Liste von sukzessiven Verfeinerung eines Grund-Dreiecksnetzes  $\mathcal{T}_1$  enthält, ergibt dann eine Sequenz von geschachtelten Finite-Elemente-Räume

$$V_1 \subset V_2 \subset \dots \subset V_J.$$

### V-Zyklus

Wir betrachten die Diskretisierungen des obigen Problems bezüglich den obigen geschachtelten Finite-Elemente-Räumen. Das Mehrgitterverfahren kann dann durch Algorithmus 1 beschrieben werden. Diesen ruft man initial mit

$$\mathbf{MG}(\mathbf{A}, \mathbf{0}_J, \mathbf{f}_J, J)$$

auf, wobei  $\mathbf{0}_J$  und  $\mathbf{f}_J$  der Null-Vektor und rechte Seite auf Level  $J$  sind.

---

**Algorithmus 1** Mehrgitterverfahren  $\mathbf{u}_j = \mathbf{MG}(\mathbf{A}, \mathbf{u}_j, \mathbf{f}_j, j)$ 

---

*Input:* Systemmatrizen  $\mathbf{A} = \{\mathbf{A}_j\}_{j=1}^L$ , Startvektor  $\mathbf{u}_j$ , rechte Seite  $\mathbf{f}_j$ , Level  $j$

*Output:* Lösungsvektor  $\mathbf{u}_j$

**if**  $j = 1$  **then**

löse die Gleichung  $\mathbf{A}_j \mathbf{u}_j = \mathbf{f}_j$  exakt, etwa mit dem CG-Verfahren

**else**

setze  $\mathbf{u}_j^{(0)} := \mathbf{u}_j$  und führe  $K$  Vorglättungsschritte aus:

$$\mathbf{u}_j^{(k)} := \mathbf{u}_j^{(k-1)} + \mathbf{B}_j^{-1}(\mathbf{f}_j - \mathbf{A}_j \mathbf{u}_j^{(k-1)}), \quad k = 1, 2, \dots, K$$

bestimme das Residuum  $\mathbf{r}_j := \mathbf{f}_j - \mathbf{A}_j \mathbf{u}_j^{(K)}$

restringiere das Residuum  $\mathbf{r}_j$  zu  $\mathbf{r}_{j-1}$

rufe  $\mathbf{c}_{j-1} := \mathbf{MG}(\mathbf{A}, \mathbf{0}, \mathbf{r}_{j-1}, j-1)$  auf

prolongiere die Korrektur  $\mathbf{c}_{j-1}$  zu  $\mathbf{c}_j$

setze  $\mathbf{u}_j^{(0)} := \mathbf{u}_j^{(K)} + \mathbf{c}_j$  und führe  $K$  Nachglättungsschritte aus:

$$\mathbf{u}_j^{(k)} := \mathbf{u}_j^{(k-1)} + \mathbf{B}_j^{-1}(\mathbf{f}_j - \mathbf{A}_j \mathbf{u}_j^{(k-1)}), \quad k = 1, 2, \dots, K$$

setze  $\mathbf{u}_j := \mathbf{u}_j^{(L)}$

**end if**

---

Da die Prolongation einfach die Multiplikation mit der jeweiligen Prolongationsmatrix und die Restriktion die Multiplikation mit der jeweiligen, transponierten Prolongationsmatrix sind, benötigt man zur Umsetzung von Algorithmus 1 noch Implementierungen von Glättern.

## Glätter

Aus der Glättungsvorschrift

$$\mathbf{u}_j^{(k)} := \mathbf{u}_j^{(k-1)} + \mathbf{B}_j^{-1}(\mathbf{f}_j - \mathbf{A}_j \mathbf{u}_j^{(k-1)}), \quad k = 1, 2, \dots, K$$

im Mehrgitteralgorithmus ergeben sich durch geeignete Wahlen von  $\mathbf{B}_j$  unterschiedliche Glätter. Für  $\mathbf{B}_j = \alpha^{-1} \vec{I}_j$  erhält man das Richardson-Verfahren, für  $\mathbf{B}_j = \mathbf{D}_j$  das Jacobi-Verfahren und für  $\mathbf{B}_j = \mathbf{D}_j + \mathbf{L}_j$  beziehungsweise  $\mathbf{B}_j = \mathbf{D}_j + \mathbf{U}_j$  das Vorwärts- bzw. Rückwärts-Gauss-Seidel-Verfahren. Hierbei wird die übliche additive Zerlegung  $\mathbf{A}_j = \mathbf{L}_j + \mathbf{D}_j + \mathbf{U}_j$  in echte untere Dreiecksmatrix, Diagonalmatrix und echte obere Dreiecksmatrix vorausgesetzt.

Im Rahmen dieser Programmierübung soll ein *symmetrischer* V-Zyklus implementiert werden. Daher ist in den Glättern stets auf Symmetrie zu achten. Während das Richardson- und das Jacobi-Verfahren per Konstruktion symmetrisch sind, muss bei der Wahl eines Gauss-Seidel-Glätters im Vorglättter die Vorwärts-Variante und im Nachglättter die Rückwärts-Variante zum Einsatz kommen.

### Aufgabe 1.

Schreiben Sie eine Funktion

```
function mgmesh = mgmesh_initialise(mesh),
```

welches ein Dreiecksnetz nimmt und das entsprechende Mehrgitter-Dreiecksnetz zurückgibt. Dabei soll `mgmesh` hier und auch nachfolgend jeweils ein Struct sein, welches  $\mathbf{P}$ ,  $\mathbf{B}$ ,  $\{\mathbf{F}_j\}_{j=1}^J$  und  $\{\mathbf{T}_j\}_{j=1}^{J-1}$  in den Feldern `mesh.P`, `mesh.B`, `mesh.F` und `mesh.T` gespeichert hat, wobei `mesh.F` und `mesh.T` Cell-Arrays sind. Die Prolongationsmatrizen  $\mathbf{T}_j$  sind dabei jeweils im `sparse`-Format zu speichern.

Implementieren sie weiter eine Funktion

```
function mgmesh = mgmesh_refine(mgmesh),
```

welche das Mehrgitter-Dreiecksnetz einmal verfeinert.

Beachten Sie die neuen, roten Korrekturen auf Blatt 1.

### Aufgabe 2.

Schreiben Sie eine Funktion

```
function g = compute_nodal_interpolation_mg(gf, mgmesh, l),
```

welche den Koeffizientenvektor  $\mathbf{g}$  der nodalen Approximation von  $g$  für das  $l$ -te Dreiecksnetz in dem übergebenen Mehrgitter-Dreiecksnetz berechnet. Die Funktion  $g$  wird dabei als `function handle gf` übergeben.

Implementieren sie weiter eine Funktion

```
function mgmesh_function_plot(g, mgmesh, l),
```

welche die durch den Koeffizientenvektor  $\mathbf{g}$  und dem  $l$ -ten Dreiecksnetz in dem übergebenen Mehrgitter-Dreiecksnetz definierte Funktion visualisiert.

### Aufgabe 3.

Schreiben Sie eine Funktion

```
function [M, A] = assemble_mass_and_stiffness_mg(mgmesh),
```

welche die Massen- und Steifigkeitsmatrizen im `sparse`-Format für alle Dreiecksnetze in dem übergebenen Mehrgitter-Dreiecksnetz berechnet und in den Cell-Arrays  $\mathbf{M}$  und  $\mathbf{A}$  zurückgibt.

Implementieren sie weiter die Modifikation der CG-Funktion

```
function u = cg_solver_mod(A, b, u, eps_tol),
```

die das übergebene  $\mathbf{u}$  als Startwert benutzt.

### Aufgabe 4.

Schreiben Sie Funktionen

```
function B = extract_smoother_richardson(A, alpha),
```

```
function B = extract_smoother_jacobi(A),
```

```
function B = extract_smoother_fGS(A),
```

```
function B = extract_smoother_bGS(A),
```

welche jeweils die definierenden Matrizen  $\{\mathbf{B}_j\}_{j=1}^J$  des jeweiligen Glätters im `sparse`-Format in dem Cell-Array  $\mathbf{B}$  zurückgeben.

### Aufgabe 5.

Schreiben Sie eine Funktion

```
function u = vcycle(u, mgmesh, A, b, Bv, Bn, j, K),
```

die einen Schritt des Mehrgitterverfahrens ausführt. Hierbei bezeichnen  $\mathbf{B}_v$  und  $\mathbf{B}_n$  Cell-Arrays der Matrizen die den Glätter definieren, wobei die ersten für die  $K$  Vor- und die zweiten für die  $K$  Nachglättungen benutzt werden.

Hinweis. Die Verwendung des Backslash-Operators für den Term  $\mathbf{B}^{-1}$  ist empfohlen, da dieser die jeweilig diagonale respektive Dreiecksstruktur von  $\mathbf{B}$  nutzt.

### Aufgabe 6.

Schreiben Sie Funktionen

```
function [us, ts] = solve_poisson_problem_cg(ff, mgmesh, eps_tol),
function [us, ts] = solve_poisson_problem_cgcascade(ff, mgmesh, eps_tol),
function [us, ts] = solve_poisson_problem_mg(ff, mgmesh, K, R, alpha),
function [us, ts] = solve_poisson_problem_mgcascade(ff, mgmesh, K, R, alpha),
```

welche das durch das Function Handle `ff` definierte Poisson-Problem auf allen Dreiecksnetzen in dem übergebenen Mehrgitter-Dreiecksnetz berechnen. Die Lösungen werden dabei als Cell-Array `us` zurückgegeben, in dem Vektor `ts` steht die für jede Lösung benötigte Laufzeit.

`solve_poisson_problem_cg` und `solve_poisson_problem_cgcascade` verwenden dabei auf jedem Netz das CG-Verfahren, wobei die erste Version als Startwert für jedes Dreiecksnetz den Nullvektor benutzt und die zweite jeweils die Prolongation der Lösung zum eins größeren Dreiecksnetzes.

`solve_poisson_problem_mg` benützt zum Lösen auf dem  $j$ -ten Dreiecksnetz  $j \cdot R$  Iterationen des V-Zyklus mit  $K$  Vor- und Nachglättungen. Als Glätter wird dabei der Richardson-Glätter zum Wert  $\alpha$  verwendet, als Startwert für jedes Dreiecksnetz der Nullvektor.

`solve_poisson_problem_mgcascade` benützt zum Lösen auf dem  $j$ -ten Dreiecksnetz  $R$  Iterationen des V-Zyklus mit  $K$  Vor- und Nachglättungen, dabei wird die Prolongation der Lösung zum eins größeren Dreiecksnetzes jeweils als Startwert benutzt.

### Aufgabe 7.

Schreiben Sie ein Skript, welches das Poisson-Problem mit der rechten Seite,

$$f\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \frac{25}{4}\pi^2 \cos(2\pi x) \cos\left(\frac{3}{2}\pi y\right),$$

auf den Dreiecksnetzen löst, die Sie durch sechsmaliges Verfeinern des durch

```
mgmesh_initialize(mesh_refine(mesh_generation_square()))
```

generierten Mehrgitter-Dreiecksnetz erhalten. Dabei verwenden Sie die Funktionen aus Aufgabe 6.

Sie wählen dazu  $\text{eps\_tol} = 10^{-8}$ ,  $K = 5$ ,  $R = 2$  und  $\alpha = 0.1$ .

Berechnen Sie den  $L^2$ - und  $H^1$ -Fehler aller numerischen Lösungen gegen die exakte Lösung,

$$u\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \cos(2\pi x) \cos\left(\frac{3}{2}\pi y\right),$$

und plotten Sie diese jeweils fuer jede der vier Methoden gegen das Verfeinerungslevel. Plotten Sie ebenso die jeweiligen Laufzeiten jeder der vier Methoden gegen das Verfeinerungslevel.

Was fällt auf?

### Aufgabe 8.

Wiederholen Sie Aufgabe 7 für den Jacobi-Glätter und den Gauss-Seidel-Glätter mit  $K = 1$  und  $K = 2$ .