Programmierblatt 3.

Abgabewoche: **18.11.–22.11.2024**

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 Ω 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 die Koeffizientenvektoren einer Finite-Elemente-Funktion von einem Netz auf ein einmal verfeinertes 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* $\{\mathbf{F}_\ell\}_{\ell=1}^{L}$ dargestellt werden. Bei einer Verfeinerung werden wie gewohnt die neuen Eckpunkte einfach hinten an die Punkteliste 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* $\{\mathbf{T}_\ell\}_{\ell=1}^{L-1}$ gespeichert. Die Prolongationsmatrix beschreibt dabei jeweils die lineare Abbildung, welche dem Koeffizientenvektor einer Finite-Elemente Funktion denjenigen Koeffizientenvektor zuordnet, der dieselbe Funktion auf dem einmal verfeinerten Netz darstellt.

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

$$V_1 \subset V_2 \subset \ldots \subset V_L.$$

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 $\mathbf{P}$, $\mathbf{B}$, $\{\mathbf{F}_\ell\}_{\ell=1}^{L}$ und $\{\mathbf{T}_\ell\}_{\ell=1}^{L-1}$ in den Feldern `mgmesh.P`, `mgmesh.B`, `mgmesh.F` und `mgmesh.T` gespeichert hat, wobei `mgmesh.F` und `mgmesh.T` Cell-Arrays sind. Die Prolongationsmatrizen $\mathbf{T}_\ell$ sind dabei jeweils im `sparse`-Format zu speichern. Weiter soll `mgmesh` ebenfalls ein Feld `mgmesh.np` haben, der als Vektor die für jede Elementliste notwendige Länge der Punktliste angibt.

Implementieren Sie weiter eine Funktion

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

welche das Mehrgitter-Dreiecksnetz einmal verfeinert.

Aufgabe 2.

Schreiben Sie eine Funktion

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

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, ll),
```

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 mgsys = mgsys_assemble(mgmesh, ff),
```

welche die Massen- und Steifigkeitsmatrizen im `sparse`-Format für alle Dreiecksnetze in dem übergebenen Mehrgitter-Dreiecksnetz berechnet und in dem Struct `mgsys` als Cell-Arrays `mgsys.M` und `mgsys.A` zurückgibt. Weiter sei in dem Struct `mgsys` ein Cell-Array `mgsys.b`, welches für alle Dreiecksnetze in dem übergebenen Mehrgitter-Dreiecksnetz die diskrete rechte Seite enthält. Die Funktion f wird dabei als `function handle ff` übergeben.

Implementieren Sie weiter eine Funktion

```
function [u, t] = mgsys_solve_backslash(mgmesh, mgsys),
```

welche die durch das Struct `mgsys` definierten diskreten Poisson-Problem auf allen Dreiecksnetzen in dem übergebenen Mehrgitter-Dreiecksnetz löst. Dabei benutzen Sie den in `MATLAB` vorhandenen \-Löser. Die Lösungen sollen als Cell-Array `u` zurückgegeben werden. Im Vektor `t` steht die für jede Lösung benötigte Laufzeit (= Zeit für den \-Löser).

Aufgabe 4.

Modifizieren Sie Ihre diagonal vorkonditionierte CG-Funktion

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

so dass sie das übergebene `u` als Startwert benutzt. Beachten Sie dabei, dass das Residuum am Anfang gemäss $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{u}_0$ berechnet wird.

Implementieren Sie weiter eine Funktion

```
function [u, t] = mgsys_solve_nested_dpcg(mgmesh, mgsys, eps_tol),
```

welche die durch das Struct `mgsys` definierten diskreten Poisson-Probleme auf allen Dreiecksnetzen in dem übergebenen Mehrgitter-Dreiecksnetz löst. Dabei benutzen Sie Ihr diagonal vorkonditioniertes CG-Verfahren, wobei Sie als Startwert für jedes Dreiecksnetz jeweils die Prolongation der Lösung zum eins gröberen Dreiecksnetzes nutzen. Die Lösungen sollen dabei als Cell-Array `u` zurückgegeben werden. Im Vektor `t` steht die für jede Lösung benötigte Laufzeit (= Zeit für die Prolongation und für das CG-Verfahren auf dem aktuellem und allen gröberen Dreiecksnetzen).

Glätter

Aus der Glättungsvorschrift

$$\mathbf{u}_\ell^{(k)} := \mathbf{u}_\ell^{(k-1)} + \mathbf{B}_\ell^{-1}\left(\mathbf{f}_\ell - \mathbf{A}_\ell \mathbf{u}_\ell^{(k-1)}\right), \quad k = 1, 2, \ldots, K,$$

im Mehrgitteralgorithmus ergeben sich durch geeignete Wahlen von $\mathbf{B}_\ell$ unterschiedliche Glätter. Für $\mathbf{B}_\ell = \alpha^{-1}\mathbf{I}_\ell$ erhält man das Richardson-Verfahren, für $\mathbf{B}_\ell = \mathbf{D}_\ell$ das Jacobi-Verfahren und für $\mathbf{B}_\ell = \mathbf{D}_\ell + \mathbf{L}_\ell$ beziehungsweise $\mathbf{B}_\ell = \mathbf{D}_\ell + \mathbf{U}_\ell$ das Vorwärts- bzw. Rückwährts-Gauss-Seidel-Verfahren. Hierbei wird die übliche additive Zerlegung $\mathbf{A}_\ell = \mathbf{L}_\ell + \mathbf{D}_\ell + \mathbf{U}_\ell$ 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 Vorglätter die Vorwärts-Variante und im Nachglätter die Rückwärts-Variante zum Einsatz kommen.

Aufgabe 5.

Schreiben Sie eine Funktion

```
function mgsmoother = mgsmoother_extract(mgsys, alpha),
```

welche die jeweiligen Matrizen $\{\mathbf{B}_j\}_{j=1}^{J}$ des jeweiligen Glätters im `sparse`-Format in den Feldern des Struct `mgsmoother` als Cell-Arrays `mgsmoother.Bv` und `mgsmoother.Bn` zurückgeben. Dabei sei `mgsmoother.Bv` der Vor- und `mgsmoother.Bn` der Nachglätter. Die Auswahl des Glätters passiere mit Hilfe des Arguments `alpha`: Für ein numerisches `alpha` geben sie den zugehörigen Richardson-Glätter zurück, für die Werte `alpha='j'` und `alpha='gs'` den Jacobi-Glätter respektive den Gauss-Seidel-Glätter.

Da die Prolongation einfach die Multiplikation mit der jeweiligen Prolongationsmatrix und die Restriktion die Multiplikation mit der jeweiligen, transponierten Prolongationsmatrix sind, ergibt sich nun der *V-Zyklus* mithilfe der Glätter durch Algorithmus 1, indem man diesen initial mit

$$\mathbf{vcycle}(\mathbf{u}_\ell, \ell, \mathbf{A}, \mathbf{f}_\ell)$$

aufruft. Dabei ist $\mathbf{u}_\ell$ die zu verbessernde Lösung und $\mathbf{f}_\ell$ die rechte Seite auf Level ℓ.

Aufgabe 6.

Schreiben Sie eine Funktion

```
function u = mgsmoother_vcycle(u, b, ll, mgmesh, mgsys, mgsmoother, K),
```

die einen V-Zyklus auf u mit K Vor- und Nachglättungen anwendet.

Hinweis. *Die Verwendung des* `\`*-Lösers für den Term* $\mathbf{B}^{-1}$ *ist empfohlen, da dieser die jeweilig diagonale respektive Dreiecksstruktur von den Glättern* $\mathbf{B}$ *ausnutzt.*

Implementieren Sie weiter eine Funktion

```
function [u, t] = mgsys_solve_nested_vcycle(mgmesh, mgsys, mgsmoother, K, J),
```

welche die durch das Struct `mgsys` definierten diskreten Poisson-Probleme auf allen Dreiecksnetzen in dem übergebenen Mehrgitter-Dreiecksnetz löst. Dabei benutzen Sie als Löser J Iterationen des V-Zykluses mit K Vor- und Nachglättungen, wobei Sie als Startwert für jedes Dreiecksnetz jeweils die Prolongation der Lösung zum eins gröberen Dreiecksnetz nutzen. Die Lösungen sollen dabei als Cell-Array u zurückgegeben werden. Im Vektor `t` steht die für jede Lösung benötigte Laufzeit (= Zeit für die Prolongation und für die J V-Zyklen, auf aktuellem und allen gröberen Dreiecksnetzen).

Algorithmus 1 V-Zyklus $\mathbf{u}_\ell = \mathbf{vcycle}(\mathbf{u}_\ell, \ell, \mathbf{A}, \mathbf{f}_\ell)$

Input: Startvektor $\mathbf{u}_\ell$, Level ℓ, ,Systemmatrizen $\mathbf{A} = \{\mathbf{A}_\ell\}_{\ell=1}^{L}$, rechte Seite $\mathbf{f}_\ell$
Output: Lösungsvektor $\mathbf{u}_\ell$

if $\ell = 1$ **then**
  löse die Gleichung $\mathbf{A}_\ell \mathbf{u}_\ell = \mathbf{f}_\ell$ exakt (mit dem \-Solver)
else
  setze $\mathbf{u}_\ell^{(0)} := \mathbf{u}_\ell$ und führe K Vorglättungsschritte aus:

$$\mathbf{u}_\ell^{(k)} := \mathbf{u}_\ell^{(k-1)} + \mathbf{B}_\ell^{-1}\left(\mathbf{f}_\ell - \mathbf{A}_\ell \mathbf{u}_\ell^{(k-1)}\right), \quad k = 1, 2, \ldots, K$$

  bestimme das Residuum $\mathbf{r}_\ell := \mathbf{f}_\ell - \mathbf{A}_\ell \mathbf{u}_\ell^{(K)}$
  restringiere das Residuum $\mathbf{r}_\ell$ zu $\mathbf{r}_{\ell-1}$
  rufe $\mathbf{c}_{\ell-1} := \mathbf{vcycle}(\mathbf{0}, \ell - 1, \mathbf{A}, \mathbf{r}_{\ell-1})$ auf
  prolongiere die Korrektur $\mathbf{c}_{\ell-1}$ zu $\mathbf{c}_\ell$
  setze $\mathbf{u}_\ell^{(0)} := \mathbf{u}_\ell^{(K)} + \mathbf{c}_\ell$ und führe K Nachglättungsschritte aus:

$$\mathbf{u}_\ell^{(k)} := \mathbf{u}_\ell^{(k-1)} + \mathbf{B}_\ell^{-1}\left(\mathbf{f}_\ell - \mathbf{A}_\ell \mathbf{u}_\ell^{(k-1)}\right), \quad k = 1, 2, \ldots, K$$

  setze $\mathbf{u}_\ell := \mathbf{u}_\ell^{(K)}$
end if

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 siebenmaliges Verfeinern des durch

```
mgmesh_initialise(mesh_refine(mesh_generation_square()))
```

generierten Mehrgitter-Dreiecksnetz erhalten. Dabei benutzen Sie alle drei Verfahren:

```
[u, t] = mgsys_solve_backslash(mgmesh, mgsys),
[u, t] = mgsys_solve_nested_dpcg(mgmesh, mgsys, eps_tol),
[u, t] = mgsys_solve_nested_vcycle(mgmesh, mgsys, mgsmoother, K, J).
```

Für das CG-Verfahren wählen sie dazu `eps_tol` $= 10^{-8}$. Bei dem Glätter wählen Sie entweder

Richardson:	`alpha` $= 0.1$,	$K = 3$,	$J = 2$;
Jacobi:	`alpha` $=$ `'j'`,	$K = 2$,	$J = 1$;
Gauss-Seidel:	`alpha` $=$ `'gs'`,	$K = 2$,	$J = 1$.

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

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

jeweils auf einem eins feineren Dreicksnetz mittels der nodalen Punktinterpolation von u und plotten Sie diese jeweils für jede der vier Methoden gegen das Verfeinerungslevel. Plotten Sie ebenso die jeweiligen Laufzeiten jeder der vier Methoden gegen das Verfeinerungslevel. Was fällt auf?