O. Junge, D. Karrasch 15. Mai 2023

Übung 8

Schreiben Sie, angelehnt an die Vorwärtssubstitution aus der Vorlesung, eine Funktion zur Rückwärtssubstition. Gegeben seien weiterhin

```
julia> A = [0 1; 1 1]
2×2 Matrix{Int64}:
 0  1
 1  1

julia> L, R, p = lu(A);
```

Nutzen Sie nun `L`, `R` und `p` geeignet, um das lineare Gleichungssystem

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} x = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

(und weitere, zufällig gewählte rechte Seiten) zu lösen. Vergleichen Sie Ihr Ergebnis mit Julias backslash-Operator.
Hinweis: Zur besseren Vergleichbarkeit empfiehlt es sich, den Code in eine Funktion zu packen:

```
function mysolve(A, b)
    L, R, p = lu(A)
    # Ihr Code
    return x
end
```

mit der numerischen Lösung `x` des Gleichungssystems als Rückgabewert.

Übung 9

Wir betrachten das lineare Gleichungssystem $Ax = b$ mit

$$A = \begin{pmatrix} 10 & 100000 \\ 1 & 2 \end{pmatrix}.$$

a) Bestimmen Sie für $p = 1, 2, \infty$ die Kondition $\kappa_p(A)$.

b) Skalieren Sie das lineare Gleichungssystem $Ax = b$ durch Multiplikation mit einer Diagonalmatrix D von links, so dass beide Zeilen von DA ungefähr gleiche ∞-Norm haben. Wie verändert sich die Kondition der resultierenden Matrix DA?

Übung 10 (Programmieraufgabe)

Implementieren Sie den Thomas-Algorithmus, einen Lösungsalgorithmus für tridiagonale lineare Gleichungssysteme.

Freiwillige Zusatzaufgabe
Implementieren Sie eine matrixfreie (im Sinne, dass kein/kaum Speicher verbraucht wird) Version der Hilbertmatrix. Dazu definieren wir einen Datentyp wie folgt:

```
struct Hilbert{T<:Rational} <: AbstractMatrix{T}
    n::Int
end
```

Damit haben wir einen Datentyp `Hilbert` erstellt, der „parametrisiert" ist über den Typenparameter `T`, der wiederum ein Untertyp des (abstrakten) Datentyps Rational ist und den Elementtypen (`eltype`) der Matrix angibt. Der Typ `Hilbert` ist ein Subtyp des abstrakten Datentyps `AbstractMatrix` mit Elementtyp `T`. Ein Objekt diesen Datentyps, welches die 5×5-Hilbertmatrix darstellen soll, erstellen wir nun via `H = Hilbert{Rational{Int}}(5)`. Für ein solches `H` kann Julia jetzt bereits selbst den `eltype` bestimmen, probieren Sie es aus. Julia muss weiterhin wissen, wie groß diese „Matrix" ist. Dies fragt man gewöhnlicherweise über die Funktion `size` ab. Diese muss nun für den neuen Datentyp überladen werden:

```
Base.size(H::Hilbert) = (H.n, H.n)
```

Zugriff auf die „Felder" eines Objekts erhält man also über die Punktnotation mit nachgestelltem Namen des Feldes. Bestimmen Sie damit also die Größe von `H`. Zuletzt wollen wir auf die Komponenten der Hilbert-„Matrix" zugreifen können à la `H[2,3]`. Diese Syntax wird intern von Julia in einen Aufruf der Funktion `getindex(H, 2, 3)` umgewandelt. Wir müssen Julia also noch erklären, wie das gehen soll, und das soll hier Ihre Aufgabe sein.[1] Probieren Sie nun aus, ob Sie einzelne Spalten, Zeilen und Submatrizen extrahieren können (etwa mit Hilfe der `:`-Indizierung) und ob Matrix-Vektor-Multiplikation „out of the box" funktioniert. Vergleichen Sie Ihre Ergebnisse mit denen aus Aufgabe 2. Übertragen Sie die Vorgehensweise auf die inverse Hilbertmatrix, und überladen Sie die `inv` (aus `Base`) derart, dass es für `Hilbert`-Objekte Objekte vom Typ (z.B.) `InverseHilbert` und umgekehrt liefert.

[1] Will man bewusst Funktionen überladen, muss man angeben, aus welchem „Modul" sie stammen. Überladen Sie also `Base.getindex(H::Hilbert, i::Int, j::Int)`.