Technische Universität München
M2 - Numerische Mathematik
Dr. Laura Scarabosio

2.1. Übung. Einführung in die Programmierung (MA 8003)

Hinweis: Alle Aufgaben können ohne Schleifen gelöst werden.

Aufgabe 2.1.1: Erzeugen Sie eine 5×5 Matrix mit Zufallswerten zwischen `-10` und `10`. Setzen Sie mit einem Befehl alle Einträge, deren Sinus negativ ist, auf `pi`.

Lösung 2.1.1: `A = -10 + 20*rand(5); A(sin(A)<0) = pi`

Aufgabe 2.1.2: Gegeben sei ein Vektor `n` mit natürlichen Zahlen, z. B. `n = 2:7:100`. Lassen Sie sich alle Einträge ausgeben, die **nicht** durch 3 teilbar sind.
Tipp: Verwenden Sie `rem` oder `mod`.

Lösung 2.1.2: `n(rem(n,3) ~= 0)` oder `l = (rem(n,3) == 0); n(~l)`

Aufgabe 2.1.3: Erzeugen Sie eine 40×40-Matrix mit Zufallswerten zwischen -0.5 und 3. Finden Sie mit einem Befehl heraus,

a) ob alle Einträge positiv sind,

b) ob alle Spalten negative Einträge haben,

c) die Position der Einträge, die größer gleich 2.5 oder kleiner -0.1,

* d) die Nummern der Zeilen, die einen negativen und einen Eintrag größer als 2.9 haben.

Überprüfen Sie Ihre Ergebnisse an Stichproben.

Lösung 2.1.3:

a) `all(R(:) > 0)`

b) `all(any(R < 0))`

c) `find(R(:) >= 2.5 | R(:) < -0.1)`

d) `[r,c] = find(any(R' < 0) & any(R' > 2.9))`

Aufgabe 2.1.4 (*): Permutationen auf $\{1 \dots n\}$ können als Permutationsmatrix (eine Matrix, die nur Nullen und Einsen enthält, mit genau einem 1-Eintrag für alle Zeilen und Spalten), oder in folgender Form dargestellt werden:

$$\begin{pmatrix} 1 & 2 & \cdots & n \\ \sigma(1) & \sigma(2) & \cdots & \sigma(n) \end{pmatrix}.$$

Im letzteren Fall reicht es, die zweite Zeile zu speichern. Finden Sie Wege, die eine in die andere Form umzuwandeln.

$$\text{z. B.} \quad P = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \quad \text{korrespondiert zu} \quad \sigma = (2, 3, 1).$$

Lösung 2.1.4: `I = eye(n); P = I(:,s);`
umgekehrt: `[s,c] = find(P==1)`

Aufgabe 2.1.5: Schreiben Sie eine *Vektorfunktion* `approxequal(x, y, tol)`, die vergleicht, ob zwei Vektoren nahezu gleich sind, also ob für alle Indizes `i` gilt $|x_i - y_i| \leq tol$. Die Rückgabe soll logisch 1 oder logisch 0 sein.
Wenn zwei Matrizen als Eingabe übergeben werden, soll die Funktion vergleichen, ob die Spalten jeweils nahezu gleich sind. Die Rückgabe in diesem Fall soll ein *Spaltenvektor* mit logisch 1 oder 0 sein.
Tipp: Hört sich aufwendiger an als es ist; kann auch anonym definiert werden.

Lösung 2.1.5:

```
function l = approxequal(x, y, tol)
  l = all(abs(x-y) <= tol);
```

Aufgabe 2.1.6: Schreiben Sie eine Funktion `addprimes(s, e)`, welche die Summe aller Primzahlen zwischen `s` und `e` zurückgibt.
Tipp: Verwenden Sie die Matlabfunktion `isprime`.

Lösung 2.1.6:

```
function summe = addprimes(s, e)
  z = s:e;
  summe = sum(z(isprime(z)));
```

Aufgabe 2.1.7: Erzeugen Sie die Abbildung $f(x, y) := (x^2y^2)/2$ als *anonyme* Matlab-Funktion. Arbeitet Ihre Funktion auch, wenn man zwei Vektoren `x` und `y` gleicher Länge als Argumente übergibt? Ändern Sie Ihre Funktion ggf. ab.

Was leistet der Befehl `vectorize`?

Lösung 2.1.7:

```
f = @(x,y) (x^2*y^2)/2;
% vectorize erzeugt eine vektorisierte Version
% von f (wenn f einfach genug ist)
fvectorString = vectorize(f)

%eval wertet diesen String aus und erzeugt eine Funtion
fvector = eval(fvectorString)

% werte vektorisierte Funktion für x=1:5 und y=1:5 aus
fvector(1:5,1:5)
```

Einfacher Alternative:

```
f=@(x,y) (x.^2.*y.^2)./2;
```

Aufgabe 2.1.8 (*): Finden Sie einen möglichst einfachen Weg, der Diagonalen einer Matrix den Wert 0 zuzuweisen. Schreiben Sie eine Funktion `function A = zero_diag(A)`, die als Eingabe eine Matrix erhält und die wie oben veränderte Matrix zurückgibt.

Testen Sie Ihre Funktion an einer zufälligen 5×5-Matrix, einer 3×6-Matrix und einer 6×3-Matrix. **Achtung:** Nur die Diagonalelemente sollen auf Null gesetzt werden!

Lösung 2.1.8:

```
function A = zero_diag(A)
  [n, m] = size(A);
  A(1:n+1:min([n*m, n*n])) = 0;
```

Alternativ:

```
function A = zero_diag(A)
  A(logical(eye(size(A)))) = 0;
```

Aufgabe 2.1.9: Schreiben Sie eine Funktion `adddigits(n)`, welche die Summe der Ziffern einer natürlichen Zahl `n` mit maximal 20 Ziffern zurückgibt. Der folgende Algorithmus leistet das Gewünschte:

a) Erzeugen Sie zuerst einen Vektor `p` mit allen 10er Potenzen von 10^0 bis 10^{20}.

b) Teilen Sie `n` durch `p` und runden Sie das Ergebnis auf die nächste ganze Zahl ab.

c) Berechnen Sie für jeden Eintrag die Division mit Rest (`help rem`) durch 10.

d) Die Summe der Reste ist nun das Ergebnis.

Lösung 2.1.9:

```
function s = adddigits(n)
  p = 10.^(0:20);
  y = floor(n./p);
  r = rem(y, 10);
  s = sum(r);
```

Aufgabe 2.1.10:

a) Schreiben Sie eine Funktion `trapez`, die den Wert des Integrals $I = \int_a^b f(x)\,\mathrm{d}x$ mit der Trapezregel berechnet ($I_T = (b-a)/2 * (f(a) + f(b))$). Die Eingabeparameter sollen die zu integrierende Funktion `f` (ein Handle) und die Intervallgrenzen `a` und `b` sein.

   Überprüfen Sie das Ergebnis Ihrer Funktion, indem Sie für f eine affin-lineare Funktion ansetzen (hier ist die Trapezregel exakt).

b) Erweitern Sie Ihre Funktion `trapez` um einen Parameter `n`. Dieser soll die Anzahl von äquidistanten Unterteilungen des Intervalls $[a, b]$ angeben, in der jeweils die Trapezregel angewendet wird. Vergleichen Sie Ihr Ergebnis für $n = 100$ mit einem Aufruf von `quadv` anhand der Funktion $f(x) = sin^2(x) \cdot x^2$ im Intervall $[-6, 6]$.

Lösung 2.1.10:

```
function [ I ] = trapez(f, a, b, n)
  h = (b-a)/n;
  l = linspace(a, b, n+1);
  w = f(l(1:end-1))+f(l(2:end));
  I = h/2*sum(w);
```