Kapitel 12

Polynome

12.1 Grundlagen

In MATLAB werden Polynome durch ihren Koeffizientenvektor repräsentiert, d.h. der Vektor `p=[p1,p2,...,pn]` stellt das Polynom,

$$p_1 x^{n-1} + p_2 x^{n-2} + p_3 x^{n-3} + \cdots + p_n \ , \tag{12.1}$$

dar. Für ein Polynom vom Grad $n-1$ braucht man daher einen Vektor der Länge n. Für die Auswertung ein solchen Polynoms für verschiedene Werte von x,

$$y_i = p_1 x_i^{n-1} + p_2 x_i^{n-2} + p_3 x_i^{n-3} + \cdots + p_n \ , \tag{12.2}$$

stellt MATLAB die Funktion `y=polyval(p,x)` zur Verfügung. Die Variable x kann dabei ein Skalar, ein Vektor, bzw. eine Matrix sein, y hat dann immmer die gleiche Größe wie x.

Will man also z.B. das Polynom

$$y_i = x_i^3 + 2x_i^2 + x_i + 3 \ , \tag{12.3}$$

darstellen, kann man Folgendes tun:

```
p = [1,2,1,3];
x = linspace(-2,2,30); y = polyval(p,x);
plot(x,y,'b');
```

Die Auswertung erfolgt natürlich mit dem Horner-Schema, das hier am Beispiel eines Polynomes dritten Grades demonstriert wird,

$$y_i = ((p_1 x + p_2)\, x + p_3)\, x + p_4 \ , \tag{12.4}$$

wobei die Anzahl der Multiplikationen pro x-Wert von $m(m+1)/2$ auf m reduziert wird, die Anzahl der Additionen bleibt mit m gleich. Daraus folgt, dass das Horner-Schema viel effizienter ist.

Es gibt auch eine Auswertung für $n \times n$ Matrizen, `polyvalm`, wobei alle Multiplikationen als Matrixmultiplikationen aufgefasst werden.

12.2 Nullstellen und charakteristische Polynome

Die Nullstellen eines Polynoms können mit dem Befehl `r=roots(p)` gefunden werden. Umgekehrt erzeugt der Befehl `p=poly(r)` das Polynom `p`, wenn `r` ein Vektor von Nullstellen ist. Für Vektoren sind `roots` und `poly` inverse Funktionen bis auf Skalierungsfaktoren, Reihenfolge der Nullstellen und Rundungsfehler.

Der Aufruf von `c=poly(A)`, wobei $\mathbf{A}$ eine $n \times n$ Matrix sein muss, liefert das charakteristische Polynom c der Matrix $\mathbf{A}$. Das charakteristische Polynom ergibt sich aus folgender Determinante

$$\det\left(\mathbf{A} - \lambda\mathbf{I}\right) \ , \tag{12.5}$$

wobei λ die Eigenwerte der Matrix $\mathbf{A}$, bzw. die Nullstellen des charakteristischen Polynoms c sind.

Dies soll am Beispiel von

$$\mathbf{A} = \begin{bmatrix} 2 & 1 \\ 2 & 3 \end{bmatrix} \ , \ \mathbf{c} = \det \begin{bmatrix} 2-\lambda & 1 \\ 2 & 3-\lambda \end{bmatrix} \ , \tag{12.6}$$

demonstriert werden, wobei sich hier als Lösung

$$\mathbf{c} = (2-\lambda)(3-\lambda) - 2 = \lambda^2 - 5\lambda + 4 \tag{12.7}$$

ergibt. Die Darstellung in MATLAB ergibt `c=[1,-5,4]` und die Nullstellen des charakteristischen Polynoms liegen bei $\lambda_{1,2} = \{1, 4\}$.

Das Pascal'sche Dreieck

$$\begin{array}{ccccccc} & & & 1 & & & \\ & & 1 & & 1 & & \\ & 1 & & 2 & & 1 & \\ 1 & & 3 & & 3 & & 1 \\ & 1 & & 4 & & 6 & & 4 & & 1 \\ 1 & & 5 & & 10 & & 10 & & 5 & & 1 \\ & 1 & & 6 & & 15 & & 20 & & 15 & & 6 & & 1 \end{array} \tag{12.8}$$

kann auch in Form der Pascal'schen Matrix, hier für $n = 4$ mit z.B. dem MATLAB-Befehl `P=pascal(4)`

$$\mathbf{P} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \\ 1 & 3 & 6 & 10 \\ 1 & 4 & 10 & 20 \end{bmatrix} \tag{12.9}$$

dargestellt werden. Das charakteristische Polynom kann nun mit dem Befehl `p=poly(P)` erzeugt werden und ergibt `[1,-29,72,-29,1]`, was folgendem Polynom entspricht

$$p(x) = x^4 - 29x^3 + 72x^2 - 29x + 1 \; . \tag{12.10}$$

Pascal'sche Matrizen haben die kuriose Eigenschaft, dass der Vektor der Koeffizienten des charakteristischen Poylnoms "palindromic" ist, d.h. er ergibt das Selbe von vorne und von hinten gelesen.

Evaluiert man das charakteristische Polynom nun im Sinne der Matrixmultiplikation, so erhält man mit `R=polyvalm(p,P)`

$$\mathbf{R} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \; , \tag{12.11}$$

d.h. die Nullmatrix. Dies ist eine Folge des Cayley-Hamilton Theorems, dass besagt das eine Matrix ihre eigene charakteristische Gleichung erfüllt.

12.3 Addition von Polynomen

MATLAB stellt keinen Befehl für die Addition von Polynomen bereit. Eine solche Routine muss man sich als kleine Übungsaufgabe selbst erstellen. Da es sich bei der Addition von Polynomen um die Addition von Vektoren unterschiedlicher Länge handelt, kann nicht einfach der Befehl `plus` verwendet werden. Man muss also vorher den kürzeren der beiden Vektoren am Beginn mit Nullen auffüllen, um die gleiche Länge bei der Verwendung von `plus` zu gewährleisten.

Die Ergänzung mit Nullen kann man durch Zusammenhängen von Vektoren erreichen. Der Befehl `zeros(1,l)` erzeugt einen Zeilenvektor der der Länge l für $l > 0$ und ein leeres Array für $l \leq 0$. Dies kann man sich in diesem Fall zu Nutze machen.

Ausserdem eignet sich dieses Beispiel bestens für die Verwendung variabler Inputlisten. Dies hat den Vorteil, dass man dann beliebig viele Polynome mit einem Aufruf addieren kann. Dafür sind die Variablen `nargin` und `varargin` bestens geeignet:

```
function p = polyadd(varargin)
p = [];
```

```
for k = 1:nargin
   p1 = varargin{k}; p1 = p1(:).';  % k-tes Polynom, Zeilenvektor
   l  = length(p); l1 = length(p1); % Längen
   p  = ...
end
```

Schön ist natürlich auch, wenn man alle führenden Nullen beseitigt, so dass maximal eine überbleibt, wenn das Ergebnis das Polynom `p=[0]` ist. Dazu kann man sich des Befehls `find` bedienen und nach dem ersten Element von p suchen, das ungleich Null ist (`min(find(p~=0))`).

12.4 Differentiation und Integration von Polynomen

Für die Differentiation von Polynomen steht der Befehl `polyder` zur Verfügung. Er kann in verschiedenen Formen verwended werden:

```
k = polyder(p)
k = polyder(a,b)
[z,n] = polyder(b,a)
```

Im zweiten Fall wird die Ableitung des Produkts der Polynome a und b berechnet und im dritten Fall erhält man den Zähler z und den Nenner n der Ableitung des Polynomquotienten b/a. Will man also z.B. die Extremwerte eines Polynoms in sortierter Reihenfolge bestimmen, kann man die x- und y-Werte der Extremwerte folgendermaßen bestimmen:

```
p   = [1,1,-2,4];
e_x = sort( roots( polyder(p) ) );
e_y = polyval( p, e_x );
```

Der Befehl `sort(x)` führt dabei die Sortierung nach der Größe von x durch.

Die Integration von Polynomen erfolgt mit dem Befehl `polyint(p)` oder `polyint(p,k)`, wobei im zweiten Fall das Skalar k als Konstante der Integration verwendet wird. Ohne Angabe von k wird dafür der Wert Null verwendet.

Will man also das Integral $\int_1^2 p(x)dx$ ausführen, kann man Folgendes machen

```
p    = [1,1,1];            u = 1; o = 2;
pint = polyint(p);
r = diff(polyval(pint,[u,o]));
```

Der Befehl `diff` führt bei einem Vektor v der Länge n die Differenzberechnung $v_{i+1} - v_i$ durch, wodurch sich ein Vektor der Länge $n-1$ ergibt.

12.5 Konvolution und Dekonvolution von Polynomen

Der Befehl `w=conv(u,v)` führt die Konvoluation der Polynome u und v aus. Algebraisch ist das das Gleiche wie die Multiplikation der Polynome, deren Koeffizienten in u und v gegeben sind. Die Multiplikation $(x+1)(x-1)$ wird also in MATLAB durchgeführt mit

```
u = [1,1]; v=[1,-1];
w=conv(u,v)                    w = [1,0,-1]
```

womit sich das Polynom $x^2 - 1$ ergibt.

Als Dekonvolution bezeichnet man die Division von Polynomen. Der MATLAB-Befehl `[q,r] = deconv(v,u)` dekonvolviert den Vektor u aus dem Vektor v, d.h. es wird der Quotient v/u gebildet und in q retourniert. Ein eventuelles Restpolynom findet sich in r wieder. Die Division von v durch u ergibt z.B.

$$\begin{aligned} v(x) &= & x^3 + 2x^2 + 3x + 4 \\ u(x) &= & x + 2 \\ q(x) = v(x)/u(x) &= & x^2 + 3 \\ r(x) &= & -2 \end{aligned} \tag{12.12}$$

was in MATLAB in folgender Form durchgeführt werden kann:

```
v = [1,2,3,4]; u = [1,2];
[q,r] = deconv(v,u);

    q = [1,0,3]     r = [0,0,0,-2]

v = conv(q,u)+r
```

Hier wurde in der letzten Zeile die Umkehroperation für Division von Polynomen gezeigt.

12.6 Fitten mit Polynomen

Im Allgemeinen bezeichnet man das Ermitteln von Funktionen, die am Besten einen gegebenen Verlauf von daten entsprechen, als Fitten der Daten. Dabei gibt man eine Modellfunktion vor, die im einfachsten Fn denall eine Gerade oder ein Polynom der Ordnung k ist. Unter der Annahme, dass die Daten in den gleich langen Vektoren x und y vorliegen, wird im sogenannten "Least Squares" Verfahren die Summe der Abstandsquadrate minimiert.

Bei Vorliegen von m Datenpunkten und unter der Annahme dass die Modellfunktion mit $p(x)$ bezeichnet wird, kann die Summe der Abstandsquadrate geschrieben werden als

$$q = \sum_{j=1}^{m} (y_j - p(x_j))^2 \stackrel{!}{=} \text{Min} \,, \tag{12.13}$$

wofür ein minimaler Wert gesucht wird.

Wenn man sich auf Polynome als Modellfunktionen beschränkt, kann man für diese Aufgabe den MATLAB-Befehl `p=polyfit(x,y,n)` verwenden, der in p die Koeffizienten des "besten" Polynoms vom Grad n, d.h. einen Vektor der Länge $n+1$, retourniert. Dieses Polynom kann dann mit `polyval` im interessanten Bereich ausgewertet werden. Diese Vorgangsweise macht natürlich nur Sinn, wenn die Modellfunktion zu den Daten "passt".