Musterlösung 1

1. MATLAB-Aufgabe **Pell–Zahlen**

a)
```
function pe=pell(n)
%% Berechnet die Pell-Zahlen.
% Input: Natuerliche Zahl n.
% Output: Pell-Zahlen f_0, ..., f_n.

% Initialisiere Null-Vektor, der die Pell-Zahlen
% enthalten soll.
pe=zeros(1,n+1);

% Anfangswerte

%%%%%%%%%%% hier der Loesungsvorschlag  %%%%%%%%%%%%
pe(1)=0;
pe(2)=1;
% Iteration

for i=3:n+1
    pe(i)=2*pe(i-1)+pe(i-2);
end
%%%%%%%%%%% hier der Loesungsvorschlag  %%%%%%%%%%%%

return
```

b)
```
>> pe = pell(9);
>> pe
pe =
     0     1     2     5    12    29    70   169   408   985
```

Die ersten 10 Pell-Zahlen lauten also: $f_0 = 0, f_1 = 1, f_2 = 2, f_3 = 5, f_4 = 12, f_5 = 29, f_6 = 70, f_7 = 169, f_8 = 408, f_9 = 985$.

c) Wir führen folgenden Code aus

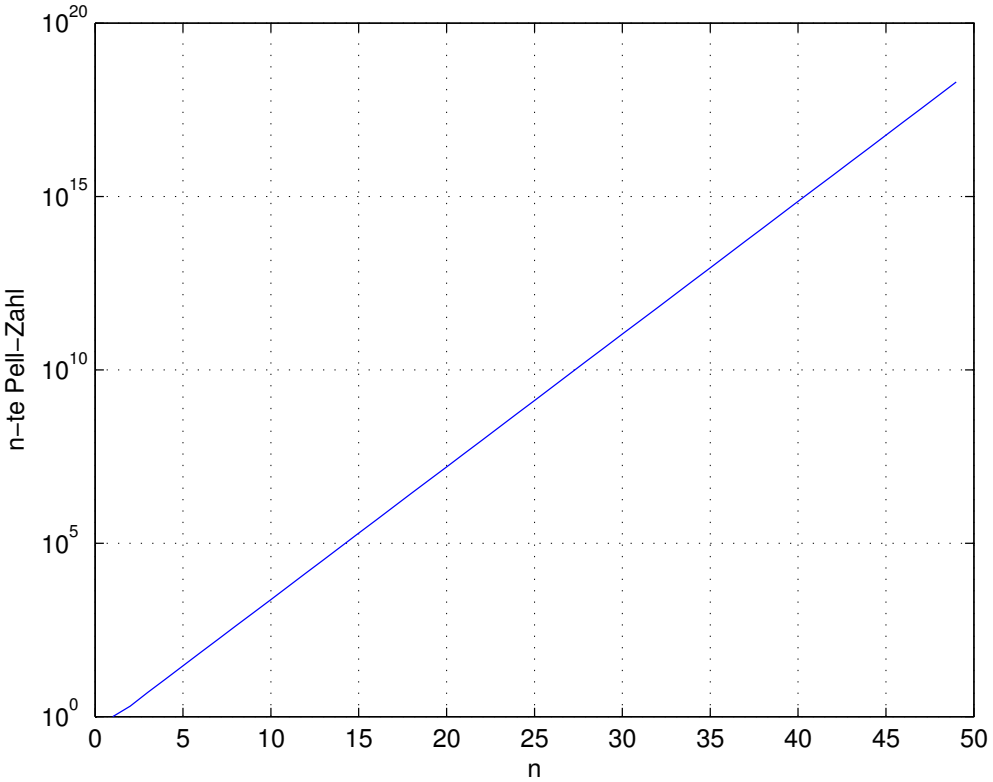

Abbildung 1: Wachstum der Pell-Zahlen

```
>> n = 50;
>> pe = pell(n-1);
>> semilogy(1:n-1, pe(2:end))
>> grid on
>> xlabel('n')
>> ylabel('n-te Pell-Zahl')
```

und erhalten die Graphik von Abb 1. Man sieht dass ausser für die Pell-Zahlen für kleines n (also insbesondere f_1, f_2, f_3) alle Pell-Zahlen auf einer Geraden zu liegen scheinen. Eine Gerade in einem Semilogplot bedeutet, dass f_n/f_{n-1} konstant ist, in unserem Fall ist `pe(end) / pe(end-1)` gleich `2.4142` und wir folgern, dass die Pell Zahlen sich für große n wie 2.4142^n verhalten.

Siehe nächstes Blatt!

2. **a)** Es gilt

$$
\begin{aligned}
F_1: &\qquad x = a + x - x^3 \Rightarrow x^3 = a \Rightarrow x = \sqrt[3]{a} \\
F_2: &\qquad x = 1 + x - \frac{x^3}{a} \overset{a\neq 0}{\Rightarrow} \frac{x^3}{a} = 1 \Rightarrow x = \sqrt[3]{a} \\
F_3: &\qquad x = \frac{1}{2}\left(x + \frac{a}{x^2}\right) \overset{x\neq 0}{\Rightarrow} x = \frac{a}{x^2} \Rightarrow x = \sqrt[3]{a}
\end{aligned}
$$

Alle Gleichungen haben also $x = \sqrt[3]{a}$ als Lösung, es sind also alles Verfahren zur bestimmung der dritten Wurzel von a. Zu bemerken ist noch, dass die Fixpunktgleichung zu F_2 für $a = 0$ und die Gleichung zu F_3 für $x = 0$ undefiniert sind. Für diese Werte ist dann auch die zugehörige Fixpunktiteration nicht anwendbar.

b)
```
function x = fixIt(f, x0, tol, kmax)
x(1) = x0;
for i=1:kmax
  x(i+1) = f(x(i));
  if abs(x(i+1) - x(i)) < tol
    break;
  end
end
```

c) Mit folgenden Matlab-Befehlen können wir die Fixpunktiteration durchführen

```
x0 = 1; tol = 1e-8; kmax = 1000; a = 3;

F1 = @(x) a+x-x^3;
x1 = fixIt(F1, x0, tol, kmax);

F2 = @(x) 1+x-x^3/a;
x2 = fixIt(F2, x0, tol, kmax);

F3 = @(x) 1/3*(2*x + a/x^2);
x3 = fixIt(F3, x0, tol, kmax);
```

die ersten fünf Iterationswerte erhalten wir mittels

```
>> x1(1:4)
           1           3         -21        9243
>> x1(5)
  -7.8966e+11
>> x2(1:5)
    1.0000    1.6667    1.1235    1.6508    1.1512
>> x3(1:5)
    1.0000    1.6667    1.4711    1.4428    1.4422
```

Bitte wenden!

Die Anzahl durchgeführter Schritte sind

```
>> length(x1)-1
    1000
>> length(x2)-1
    1000
>> length(x3)-1
     7
```

d) Für $a = 2$ und $x_0 = 1$ konvergieren die Fixpunkiterationen F_1 und F_2 also nicht. Für $a = 2$ und $x_0 = 1$ scheint die Fixpunkiterationen F_3 zu konvergieren.

e) Nur für die Iteration zu F_3 lässt sich also eine Konvergenzordnung bestimmen. Wie in Aufgabe 3 berechnen wir für F_3

```
>> e3 = abs(x3 - 3^(1/3));
p3 = [];
for k = 1:5
  p3(k) = log( e3(k+1) / e3(k+2) ) / log( e3(k) / e3(k+1) );
end
```

und erhalten als Schätzer für die Konvergenzordnung die Werte:

```
>> p3
p3 =
    3.0234    1.9200    1.9934    1.9992       Inf
```

Wir vermuten also, dass die Konvergenzordnung $p \approx 2$ ist, also dass das Verfahren quadratisch konvergiert.

3. Erst laden wir die Datei mittels

```
>> konvergenzstudie
```

in Matlab, dieses lädt den Vektor e und dann führen wir folgende Operationen aus

```
p = [];
for k = 1:5
  p(k) = log( e(k+1) / e(k+2) ) / log( e(k) / e(k+1) );
end
p
```

und erhalten

Siehe nächstes Blatt!

```
4.979058149289701
4.996796260794513
5.000148897783778
5.000160406851736
5.000028710624543
```

wir vermuten daher, dass die Konvergenzordnung $p = 5$ ist. Die leichten Schwankungen kommen daher, dass die Konvergenzordnung nur asymptotisch $p = 5$ ist, also möglicherweise erst sehr nahe bei dem Fixpunkt. Zusätzlich treten Probleme auf, weil Matlabnur mit "Computerzahlen" rechnet und diese nur beschränkt viele Stellen haben. Mit diesen Effekten ist in der Praxis immer zu rechnen.

4. **a)** Das Wilkinson Polynom ist definiert durch

$$w(x) = \prod_{i=1}^{20}(x-i) = (x-1)(x-2)\cdots(x-20).$$

Es hat die Nullstellen $1, 2, 3, \ldots, 20$.

b) Wir approximieren die Nullstellen mit Matlab und dem Befehl

```
>> format long
>> r = roots(poly(1:20))
r =
  20.000324878101402
  18.997159990805148
  18.011221676102622
  16.971132243131219
  16.048274533412592
  14.935355760260174
  14.065272732694492
  12.949055715498519
  12.033449121964885
  10.984041283443625
  10.006059681252784
   8.998394492431256
   8.000284343435283
   6.999973481009383
   5.999999755869895
   5.000000341909659
   3.999999967630562
   3.000000001049189
```

Bitte wenden!

```
   1.999999999997379
   0.999999999999841
```

c) Der Fehler zu den exakten Nullstellen berechnet sich als

```
>> r - (20:-1:1)'
ans =
   0.000324878101402
  -0.002840009194852
   0.011221676102622
  -0.028867756868781
   0.048274533412592
  -0.064644239739826
   0.065272732694492
  -0.050944284501481
   0.033449121964885
  -0.015958716556375
   0.006059681252784
  -0.001605507568744
   0.000284343435283
  -0.000026518990617
  -0.000000244130105
   0.000000341909659
  -0.000000032369438
   0.000000001049189
  -0.000000000002621
  -0.000000000000159
```

Der Fehler für die Nullstelle $x = 1$ ist sehr klein (10^{-13}) aber der Fehler für die Nullstellen $x = 14$ und $x = 15$ ist deutlich größer (10^{-2}). Numerische Verfahren haben große Probleme die Nullstellen von Polynomen hoher Ordnung zu genau zu approximieren. Dies liegt nicht an einem Bug in Matlab sondern daran, dass auf dem Computer Zahlen nicht exakt sondern als Computerzahlen mit nur einer festen Anzahl von gültigen Stellen gespeichert werden.