Modellierung und Simulation **Übung 03**
WS 2016/17 **Abgabefrist: Siehe elearning bzw. exchange**

Name: _____ **Aufwand (h):** _____

Punkte: _____

Aufgabe 1 (3 + 3 + 2 = 8 Pkt): Modellierung einer Mondlandung

(a) Modellieren Sie in SIMULINK die Landung einer Landefähre, wie sie in den Unterlagen zur Vorlesung (idealisiert und vereinfacht) formuliert ist.
Gehen Sie von einer anfänglichen Sinkgeschwindigkeit von 10 m/s und einer Ausgangshöhe von 1000 m aus; die Bremskraftkoeffizienten der beiden verfügbaren Bremsstufen seien 42,000 und 34,000, und die Masse der Landefähre 20,000 kg.
Ermitteln Sie durch simulationsgesteuerte Optimierung optimale Start- und End-Zeitpunkte für die Bremsphasen; dokumentieren Sie Ihre optimalen Ergebnisse in Bezug auf Landezeitpunkt und Geschwindigkeit beim Aufsetzen.

(b) Wirken sich Änderungen am Solver bei der Simulation Ihres Modells signifikant aus? Dokumentieren Sie Veränderungen durch die Verwendung von anderen Integrationsmethoden und veränderten Schrittweiteneinstellungen. Wie erklären Sie sich dieses Ergebnis?

(c) Nehmen Sie an, daß durch die Bremsung die Masse der Landefähre verringert wird (mit Faktor 0.001). Adaptieren Sie Ihr Modell entsprechend und dokumentieren Sie Auswirkungen in Bezug auf die Landung Ihrer Landefähre.
Ermitteln Sie erneut optimale Bremsparameter, und dokumentieren Sie Ihre neue Landezeit und Aufsetzgeschwindigkeit.

Aufgabe 2 (8 + 4 + 4 = 16 Pkt): Optimierung von Simulationsmodellen mit ES

(a) Implementieren Sie in MATLAB einen Optimierungsalgorithmus basierend auf einer Evolutionsstrategie, der die Simulation der Mondlandung aus Aufgabe 1 optimiert.

(b) Dokumentieren Sie Ihre Implementierung sowie Ihre Überlegungen zu den folgenden Fragen:
- Was muß *optimiert* werden, was ist eine geeignete Fitnessfunktion?
- Welche Parameter können *modifiziert* werden?
- Warum gerade soll gerade eine Evolutionsstrategie verwendet werden und nicht z.B. ein genetischer Algorithmus, Tabu Suche, oder Ant Colony Optimization?

(c) Welche algorithmischen Parametersetzungen führen zu eher guten, welche zu eher schlechten Ergebnissen? Wie bewerten Sie die Performance eines solchen Optimierungslaufes?
Wirkt sich die Anpassung der Mutationsweite aus?
Dokumentieren Sie Ihre Ergebnisse mit unterschiedlichen Parameter-Settings, zeigen Sie Statistiken (Graphiken, Tabellen, …).

Hinweise: Geben Sie Ihre Ausarbeitung gedruckt auf Papier ab.
Abgegebene Beispiele müssen in der Übungsstunde präsentiert werden können.

1 Modellierung einer Mondlandung

1.1 Simulink Modell der Mondlandung

Die Abbildung 1 zeigt das implementierte Modell der Mondlandung.

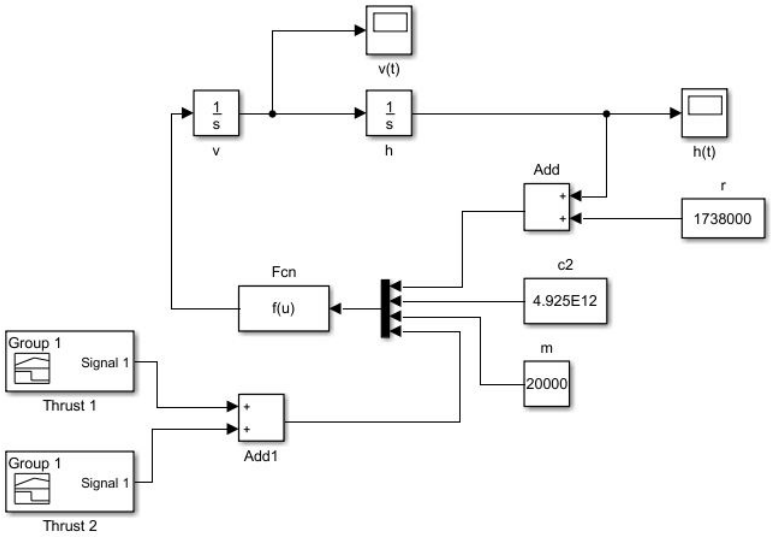

Abbildung 1: Simulink Modell der Mondlandung

1.1.1 Test 1

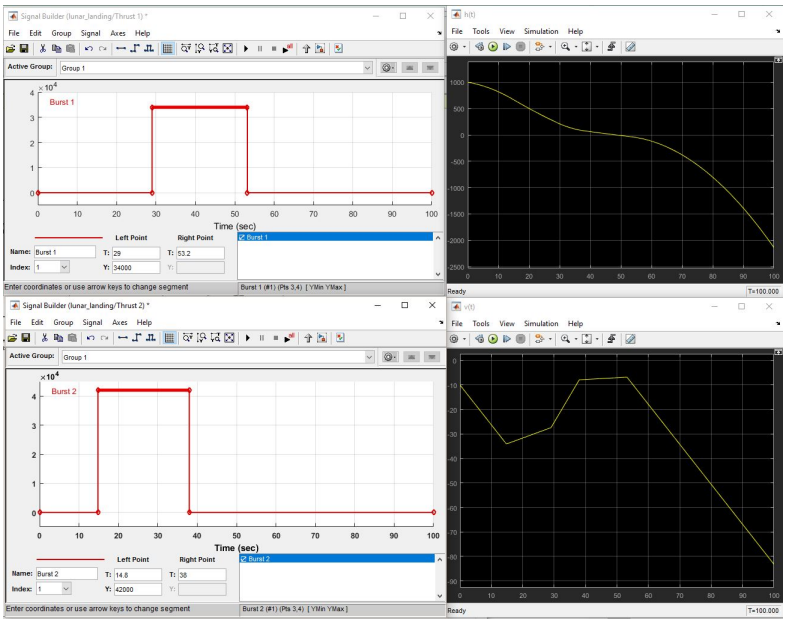

Abbildung 2: $h_{(t)}$= -0.0088 m, $v_{(t)}$= -7.17 m/s

1.1.2 Test 2

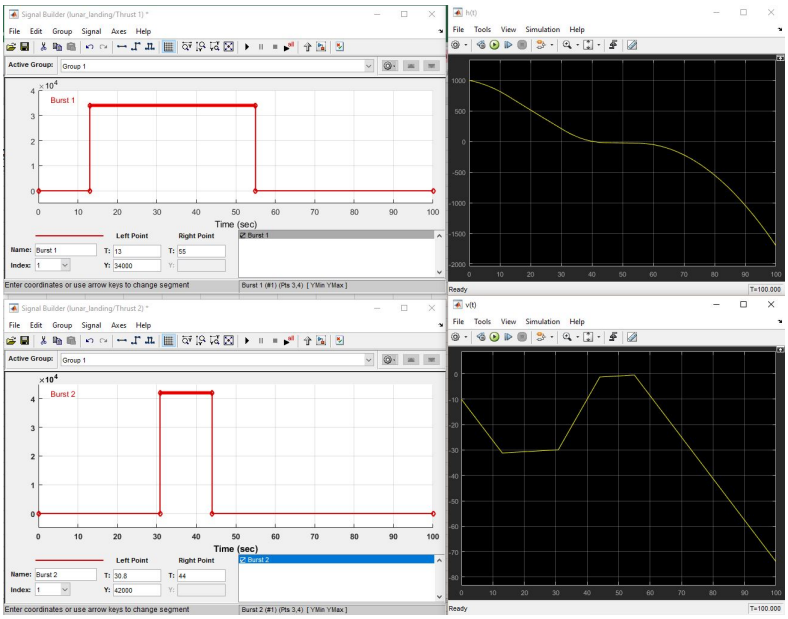

Abbildung 3: $h_{(t)}$= -0.0256 m, $v_{(t)}$= -8.72 m/s

1.1.3 Test 3

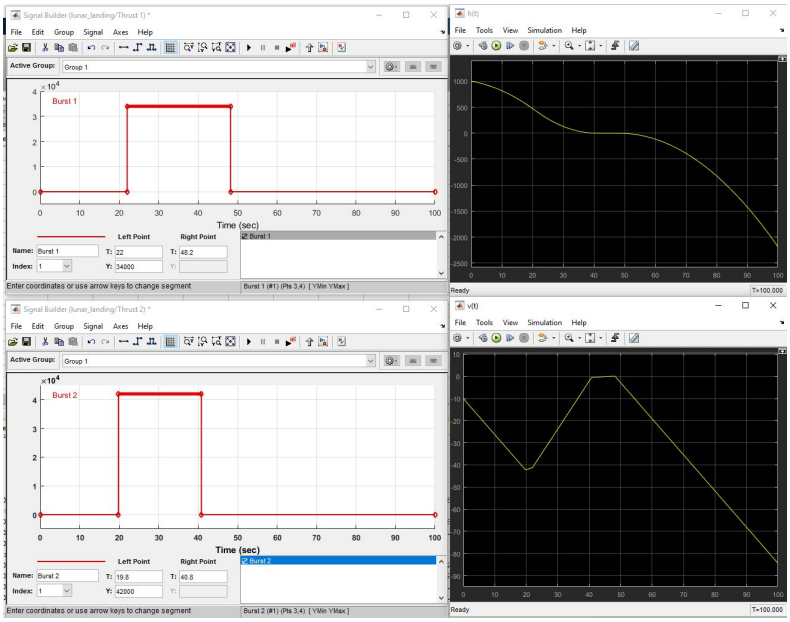

Abbildung 4: $h_{(t)}$= -0.0018 m, $v_{(t)}$= -1.44 m/s

1.1.4 Testergebnisse

Nr.	Start Thrust 1	End Thrust 1	Start Thrust 2	End Thrust 2	h(t)	v(t)
1	29	53.2	14.8	38	-0.008801562	-7.17180995
2	13	55	30.8	44	-0.02586503	-8.71786497
3	22	48.8	19.8	40.8	0.001799328	-1.441021505

Abbildung 5: Tabelle mit den Gesamtergebnissen

1.2 Solver Modifikationen

Als Beispiel wird der Lösungskandidat aus Abschnitt 1.1.4 herangezogen, mit dem gezeigt wird, wie sich die Änderungen an den Einstellungen auswirken.

1.2.1 Test Schrittweite 1 und 0.01

Nr.	Start Thrust 1	End Thrust 1	Start Thrust 2	End Thrust 2	Änderung	h(t)	v(t)
1	22	48.8	19.8	40.8	step=0.01	0.001799328	-1.441021505
					step=0.1	-0.005784244	-0.313609826
					step=1	-1.922308429	-9.593092329

Abbildung 6: Testergebnisse mit den drei Schrittweiten 0.01, 0.1 und 1

Die gravierenden Unterschiede entstehen, da mit einer zu großen Schrittweite schnelle Veränderungen im System nicht erkannt werden können. Bei schnellen Veränderungen im System ist eine kleine Schrittweite von Vorteil, bei langsamen Veränderungen eine große Schrittweite.

1.2.2 Test verschiedener Integrationsmethoden

Nr.	Start Thrust 1	End Thrust 1	Start Thrust 2	End Thrust 2	Änderung	h(t)	v(t)
1	22	48.8	19.8	40.8	ode5	0.001799328	-1.441021505
					euler	-0.0021843	-1.6195
					fixed auto	-0.0021854	-1.5762
					variable auto	-0.0021856	-0.4263

Abbildung 7: Testergebnisse mit den vier Integrationsmethoden

Die verschiedenen Integrationsmethoden *ode5*, *euler* unterscheiden sich durch ihren lokalen und globalen Error. Diese Error Indikatoren werden durch verschiedene Mechanismen die bei der Integration angewandt werden, wie gewichtetes Mittel, Miteinbeziehung von vorherigen und/oder geschätzten Nachfolgern und der Verwendung empirischer Faktoren, beeinflusst.

Listing 1: Testprogramm für die Simulation mit modifiziertem Solver

```
% Prepare model parameters
fileName    = 'parametrized_lunar_landing_modified';
modelParams = [1 22 48.2 19.8 40.8];
simParams   = simget(fileName);

% ------------------------------------------------------------------------
% test euler
% ------------------------------------------------------------------------
simParams.Solver = 'euler';
[T,X] = sim(fileName,100,simParams,modelParams);

h    = X.signals(1).values;
minHEuler = min(h);
impactIdx        = find(h < 0);
qualityEuler = X.signals(2).values(impactIdx(1));
% ------------------------------------------------------------------------

% ------------------------------------------------------------------------
% Test fix step
% ------------------------------------------------------------------------
simParams.Solver = 'FixedStepAuto';
[T,X] = sim(fileName,100,simParams,modelParams);

h    = X.signals(1).values;
minHAuto = min(h);
impactIdx   = find(h < 0);
qualityAuto = X.signals(2).values(impactIdx(1));
% ------------------------------------------------------------------------

% ------------------------------------------------------------------------
% Test fix step
% ------------------------------------------------------------------------
simParams.Solver = 'VariableStepAuto';
[T,X] = sim(fileName,100,simParams,modelParams);

h    = X.signals(1).values;
minHVAuto = min(h);
impactIdx   = find(h < 0);
qualityVAuto = X.signals(2).values(impactIdx(1));
% ------------------------------------------------------------------------
```

1.3 Modifikation der Mondlandungsmodell

Die Abbildung 8 zeigt das modifizierte Modell der Mondlandung.

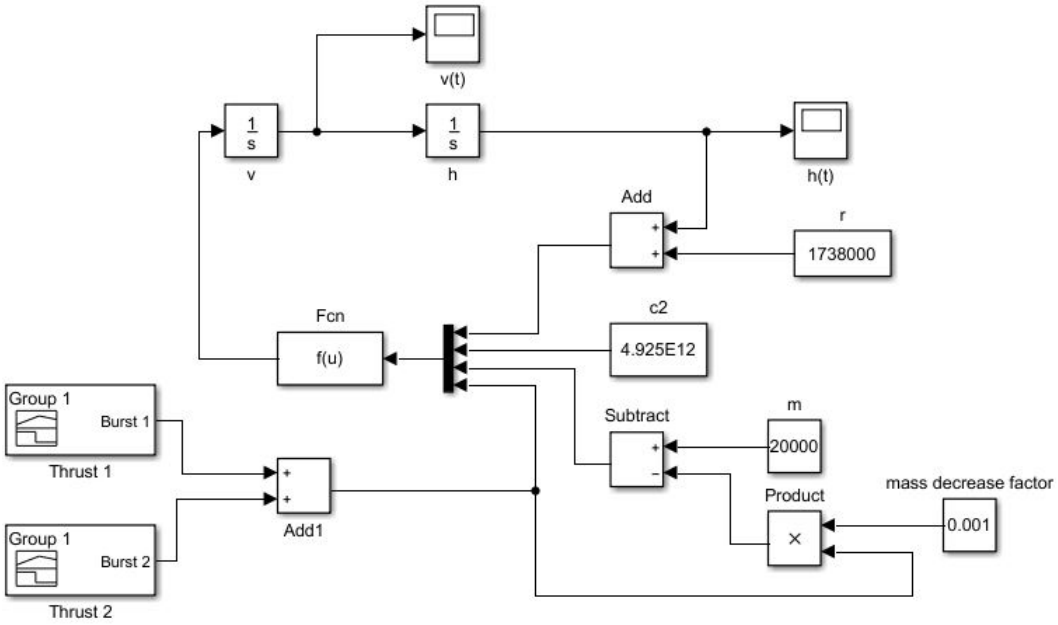

Abbildung 8: Mondladungsmodell mit Massenreduktion beim Bremsen

1.3.1 Test 1

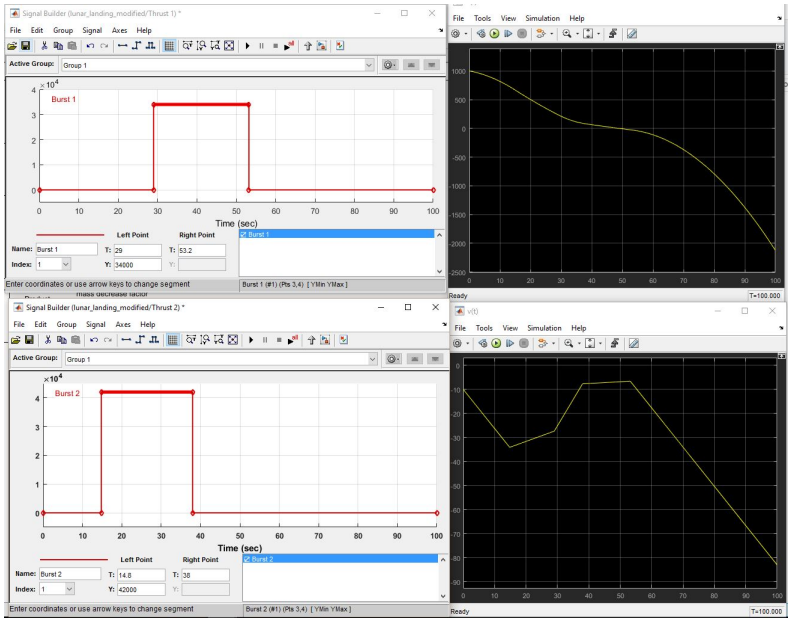

Abbildung 9: $h_{(t)}$= -0.0685 m, $v_{(t)}$= -6.91 m/s

1.3.2 Test 2

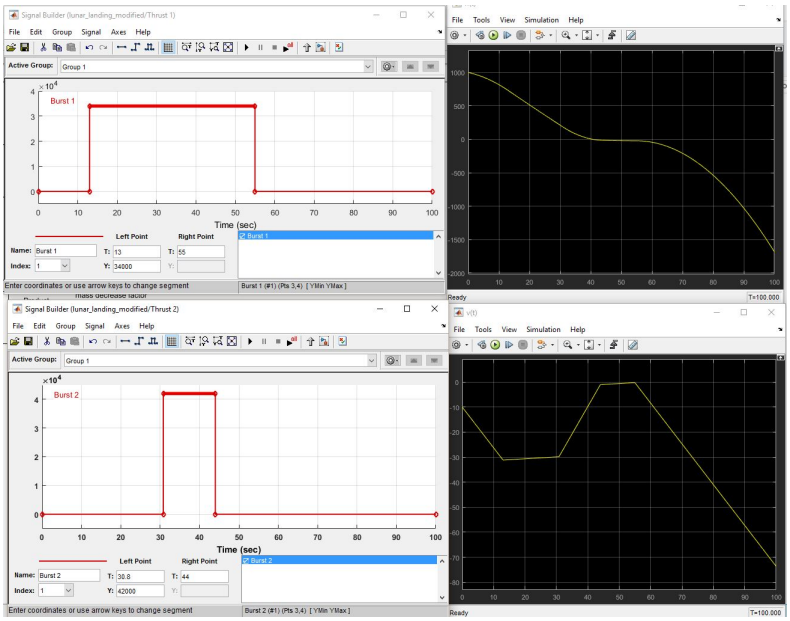

Abbildung 10: $h_{(t)}$= -0.033 m, $v_{(t)}$= -8.088 m/s

1.3.3 Test 3

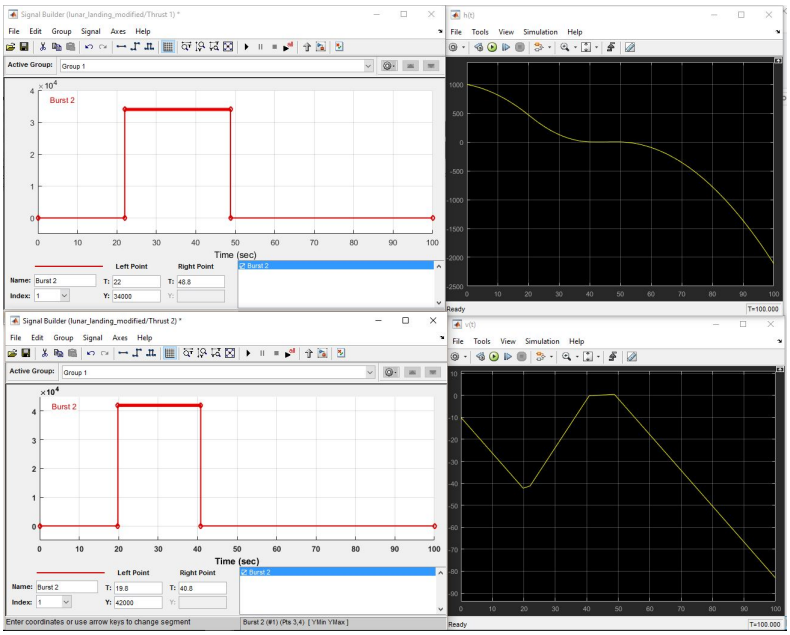

Abbildung 11: $h_{(t)}$= -0.0088 m, $v_{(t)}$= -3.4123 m/s

1.3.4 Testergebnisse

Nr.	Start Thrust 1	End Thrust 1	Start Thrust 2	End Thrust 2	h(t)	v(t)	h(t)	v(t)	h(t) div	v(t) div
1	29	53.2	14.8	38	-0.06852981	-6.908718221	-0.008801562	-7.17180995	0.059728247	-0.2630917
2	13	55	30.8	44	-0.032871106	-8.08786718	-0.02586503	-8.71786497	0.007006076	-0.6299978
3	22	48.8	19.8	40.8	-0.008805211	-3.412267071	0.001799328	-1.441021505	0.010604539	1.97124557

Abbildung 12: Tabelle mit den Gesamtergebnissen

Es wurden dieselben Parametervektoren wie für das originale Modell verwendet, wobei sich die Resultate für die ersten beiden Testfälle verbessert und für den dritten Testfall akzeptabel verschlechtert haben.

2 Optimierung mit einen Evolutionsalgorithmus

2.1 Der Evolutionsalgorithmus

Folgende Quelltexte sind das implementierte Hauptprogramm und die implementierten Funktionen des Evolutionsalgorithmus.

Listing 2: moonLanding

```
% ------------------------------------------------
% The main program with the evolution algorithm
% S1610454013, Thomas Herzog
% ------------------------------------------------
% Prepare the algorithm constants
% ------------------------------------------------
cfg                   = [];
cfg.timeSpan          = 100;
cfg.mu                = 5;
cfg.lambda            = 10;
cfg.delta             = 10;
cfg.punish            = 0.1;
cfg.deltaStep         = 1;
cfg.simFile           = 'parametrized_lunar_landing';
cfg.simParams         = simget(cfg.simFile);
cfg.maxValue          = 100;
cfg.maxUnsuccessCount = 50;
cfg.maxGenerations    = 50;

% ------------------------------------------------
% Prepare main programm members
% ------------------------------------------------
runCount          = 1;
run               = 1;
generationCount   = 1;
curUnsuccessCount = 1;
curBestQuality    = 1000;
bestQualities     = [];
bestSolutions     = [];
meanQualities     = [];
population        = [];
initialVektor     = [];

% initial candiadte
initialVektor.start1 = 14;
initialVektor.end1   = 20;
initialVektor.start2 = 10;
initialVektor.end2   = 15;
population{1}        = initialVektor;

% ------------------------------------------------
% Main loop
% ------------------------------------------------
while (run == 1)
    result        = [];
    newGeneration = [];
    qualities     = [];
    solutions     = [];

    % 1. Select
    for i=1:cfg.mu
        idxSize          = size(population);
        % break if no further mutant available
        if idxSize(:,2) == 0
```

```
55              break;
56          end
57
58          % random selection
59          randMatrix       = randperm(idxSize(:,2));
60          idx              = randMatrix(1:1);
61          solution         = population{idx};
62          population(:,idx) = [];
63
64          % 2. Bread, 3. Mutate
65          result        = bread(solution, cfg, (i - 1));
66          newGeneration = [newGeneration result.population];
67          qualities     = [qualities; result.qualities];
68          solutions     = [solutions result.population];
69      end
70
71      % Selektion of the mu best
72      qualities = sortrows(qualities);
73      population     = [];
74      for i=1:1:cfg.mu
75        population{i} = newGeneration{qualities(i,2)};
76      end
77
78      % Filll memory and modify delta depending on qualtity
79      bestQuality                = qualities(1,1);
80      meanQuality                = mean(qualities);
81      sizeQualities              = size(qualities);
82      bestQualities(runCount,1)  = bestQuality;
83      bestQualities(runCount,2)  = runCount;
84      bestSolutions{runCount}    = solutions{qualities(1,2)};
85      meanQualities(runCount,1)  = meanQuality(1,1);
86      meanQualities(runCount,2)  = runCount;
87      betterCounter              = 0;
88      for i=1:1:sizeQualities(1,2)
89          if curBestQuality < bestQuality
90              break;
91          end
92          betterCounter = betterCounter + 1;
93      end
94      % new best quality found
95      if curBestQuality > bestQuality
96          curBestQuality  = bestQuality;
97          % If we found a new best decrease unseuccess counter
98          if curUnsuccessCount > 0
99              curUnsuccessCount = curUnsuccessCount - 1;
100         end
101     % If we didn't found a new best decrease unseuccess counter
102     else
103         curUnsuccessCount = curUnsuccessCount + 1;
104     end
105     % decrease step because quality is getting better
106     if betterCounter >= (sizeQualities(1,1) / 5)
107         cfg.delta = cfg.delta - cfg.deltaStep;
108     % increase step because quality is getting worse
109     else
110         cfg.delta = cfg.delta + cfg.deltaStep;
111     end
112
113     oldBestQuality = curBestQuality;
114     % Determine if continue
115     run             = ((generationCount < cfg.maxGenerations) && (curUnsuccessCount <
    ↪  cfg.maxUnsuccessCount));
116     generationCount = generationCount + 1;
```

```
117     runCount        = runCount + 1;
118 end
119 
120 % ------------------------------------------------
121 % Plot the results
122 % ------------------------------------------------
123 sortedBestQualities  = sortrows(bestQualities);
124 sortedMeanQualities  = sortrows(meanQualities);
125 
126 runBestQuality       = sortedBestQualities(1,1);
127 runMeanQuality       = sortedMeanQualities(1,1);
128 runBestSolution      = bestSolutions{sortedBestQualities(1,2)};
129 
130 sizeBestQualities    = size(bestQualities);
131 figure;
132 xAxis                = 1:(sizeBestQualities(1,1));
133 yAxisBest            = bestQualities(:,1);
134 yAxisMean            = meanQualities(:,1);
135 plot(xAxis, yAxisBest, xAxis, yAxisMean);
136 hold on
137 xlabel('Generation');
138 ylabel('quality');
139 legend({'beat quality/generation','mean quality/generation'})
```

Listing 3: initialize.m

```
1  function solution = initialize(solution)
2  % Initializes the solution object
3  % @param solution the solution to inititalize
4  
5  % First Thrust
6  solution.start1 = rand * 100;
7  solution.end1   = rand * 100;
8  % Ensure that end is after start
9  while solution.end1 <= solution.start1
10     solution.end1 = rand * 100;
11 end
12 
13 % Second Thrust
14 solution.start2 = rand * 100;
15 solution.end2   = rand * 100;
16 % Ensure that end is after start
17 while solution.end2 <= solution.start2
18     solution.end2 = rand * 100;
19 end
20 
21 % Reset evaluation values
22 solution.quality          = [];
23 solution.heightProgress   = [];
24 solution.velocityProgress = [];
25 
26 end
```

Listing 4: bread.m

```
1  function result = bread(solution, cfg, offset)
2  % Breads the mutants for the given candidate
3  % @param solution the solution to bread mutants for
4  % @param cfg      the config holding the lambda
5  % @param offset   the offset for the solution index because merged outside
```

```

result      = [];
population  = [];
qualities   = [];

% generate lambda mutants for the given candidate
for i=1:cfg.lambda
    idx            = i + (cfg.lambda * offset);
    tmpSolution    = mutate(solution, cfg);
    tmpSolution    = evaluate(tmpSolution, cfg);
    population{i}  = tmpSolution;
    qualities(i,:) = [tmpSolution.quality idx];
end

% build result
result.population = population;
result.qualities  = qualities;

end
```

Listing 5: mutate.m

```
function solution = mutate(solution, cfg)
% Mutates the given candidate
% @param solution the solution to mutate
% @param cfg      the configuration holding configurable constants

% Mutate first thrust
tmpStart1 = mod(solution.start1 + (rand * cfg.delta), 100);
if tmpStart1 > cfg.maxValue
    solution.start1 = rand * cfg.delta;
else
    solution.start1 = tmpStart1;
end
tmpEnd1    = mod(solution.end1 + (rand * cfg.delta), 100);
if tmpEnd1 > cfg.maxValue
    solution.end1 = rand * cfg.delta;
else
    solution.end1 = tmpEnd1;
end
while solution.end1 <= solution.start1
    solution.end1 = solution.end1 + (rand * cfg.delta);
end

% Mutate second thrust
tmpStart2 = mod(solution.start2 + (rand * cfg.delta), 100);
if tmpStart2 > cfg.maxValue
    solution.start2 = rand * cfg.delta;
else
    solution.start2 = tmpStart2;
end
tmpEnd2   = mod(solution.end2 + (rand * cfg.delta), 100);
if tmpEnd2 > cfg.maxValue
    solution.end2 = rand * cfg.delta;
else
    solution.end2 = tmpEnd2;
end
while solution.end2 <= solution.start2
    solution.end2 = solution.end2 + (rand * cfg.delta);
end

% Reset evaluation members
```

```
41 solution.quality          = [];
42 solution.heightPRogress   = [];
43 solution.velocityProgress = [];
44
45 end
```

Listing 6: evaluate.m

```
1  function solution = evaluate(solution, cfg)
2  % Evaluates the simulation for the given candidate
3  % @param solution the solution to evaluate
4  % @param cfg      the configuration holding configurable constants
5
6  % Prepare model parameters
7  modelParams = [1, solution.start1, solution.end1,solution.start2, solution.end2];
8  % run simulation
9  [T,X] = sim(cfg.simFile,100,cfg.simParams,modelParams);
10
11 h                          = X.signals(1).values;
12 minH                       = min(h);
13 impactIdx                  = find(h == minH);
14 idx                        = impactIdx(1);
15 solution.heightProgress   = X.signals(1).values(idx);
16 solution.velocityProgress = -X.signals(2).values(idx);
17
18 % we haven't fully landed
19 if minH > 0
20     solution.quality = (minH/10) + cfg.punish;
21 % We have landed already
22 else
23     solution.quality = solution.velocityProgress;
24 end
25
26 end
```

2.2 Optimierungen des Evolutionsalgorithmus

2.2.1 Implementierung

Die Aufteilung der Aufgaben des Algorithmus wurden wie ind er Übung bereits gemacht übernommen. Ein Lösungskandidat wird mit fac_{punish} bestraft wenn $minH > 0$, da hier der Boden nicht erreicht wurde und daher dieser Lösungskandidat nicht zu gut bewertet werden darf. Wenn 1/5 der Lösungskandidaten bessere Lösungen gefunden hat als die letzte Generation, dann $\delta = \delta - \delta_{step}$ ansonsten $\delta = \delta + \delta_{step}$. Damit wird die Schrittweite zusätzlich erhöht oder verkleinert.

Am ende des Programms werden die besten und der Durchschnitt der Qualitäten je Generation in einem Chart angezeigt, um so den Verlauf der Suche der Lösungskandidaten zu veranschaulichen.

2.2.2 Was muss optimiert werden ?

Es müssen Werte für die Parameter des Algorithmus wie z.B μ, λ, δ und δ_{step} gefunden werden, mit denen die Wahrscheinlichkeit hoch ist gute Lösungskandidaten zu bekommen. Prinzipiell muss der Algorithmus von der konkreten Problemstellung nichts wissen, jedoch spielt es eine Rolle wie der Lösungsraum vom Algorithmus durchsucht wird und ob mit diesem Verhalten gute Lösungskandidaten zu ermitteln sind.

Aufgrund der möglichen langen Laufzeit sollte auch Parallelisierung in Betracht gezogen werden, was in dieser Übung nicht angewandt wurde.

2.2.3 Was ist eine geeignete Fitnessfunktion ?

Eine gute Fitnessfunktion ist eine Funktion, die in der Lage ist, die Lösungskandidaten je nachdem ob sie gute oder schlechte sind entsprechend zu bewerten. Die Bewertung wird im Algorithmus dazu verwendet um zu ermitteln wie hoch der Anteil an guten Lösungskandidaten war und aus welchen Lösungskandidaten eine neue Generation erzeugt wird.

2.2.4 Welche Parameter können modifiziert werden ?

Folgende Auflistung zeigt die Parameter, die modifiziert werden können:

- μ ist die Anzahl der Elter aus denen in einer Generation Mutanten erzeugt werden.
- λ ist die Anzahl der zu erzeugenden Mutanten pro Elter pro Generation
- δ ist die maximale Schrittweite, um die sich ein Parameter in einem Parametervektor pro Mutation ändern darf.
- δ_{step} ist das Delta, mit dem die Schrittweite pro Generation zusätzlich vergrößert wird, je nachdem ob eine neue beste Lösung gefunden wurde oder nicht.
- $maxUnsuccessCount$ ist die maximale Anzahl von nicht neu gefunden besten Lösungskandidaten pro Generation.
- $maxGenerationCount$ ist die maximale Anzahl von Generationen, die der Algorithmus durchläuft.
- $initialVektor$ ist der initiale Parametervektor, mit dem der Algorithmus gestartet wird.

2.2.5 Warum ein Evolutionsalgorithmus ?

Im Gegensatz zu einem *Particle Swarm* Algorithmus muss ein Evolutionsalgorithmus den Lösungsraum nicht kennen und der Lösungsraum muss kein Koordinatensystem sein. Ebenso ist ein *Particle Swarm* Algorithmus nicht anwendbar auf Optimierungsprobleme, was aber eine Mondlandung ist. Im Gegensatz zu einem *Tabu* Algorithmus benötigt ein Evolutionsalgorithmus kein Gedächtnis. Ein Evolutionsalgorithmus ist auch an sich schon dafür geeignet reelwertige Vektoren zu optimieren und kann auf Optimierungsprobleme angewendet werden.

Die Annäherung an gute Lösungskandidaten erfolgt schrittweise pro Generation in dem die besten Lösungskandidaten pro Generation herangezogen werden, um die nächste Generation zu erzeugen. Die Schrittweite um die sich die Werte eines Parametervektors pro Mutation ändern können, wird beeinflusst ob man einen guten Lösungskandidaten nahe ist oder nicht (*pro Generation eine neuer bester Lösungskandidat*).

2.3 Testfälle mit verschiedenen Konfigurationen

Die folgenden Test stellen verschiedene Konfigurationen gegenüber wie λ, δ, δ_{step} und die Anzahl der maximalen Generationen und der maximalen fehlerhaften Durchläufe. Im Chart wird die beste Konfiguration pro Generation mit ihrer Qualität angezeigt. Gibt es keine Veränderung zwischen zwei Generationen, so hat es auch keinen neuen besten Lösungskandidaten in dieser Generation gegeben. Für die genau Konfiguration eines Tests siehe bitte die Abbildung.

2.3.1 Test 1

```
% ------------------------------------------------
% Prepare the algorithm constants
% ------------------------------------------------
cfg                  = [];
cfg.timeSpan         = 100;
cfg.mu               = 5;
cfg.lambda           = 10;
cfg.delta            = 10;
cfg.punish           = 0.1;
cfg.deltaStep        = 1;
cfg.simFile          = 'parametrized_lunar_landing';
cfg.simParams        = simget(cfg.simFile);
cfg.maxValue         = 100;
cfg.maxUnsuccessCount = 50;
cfg.maxGenerations   = 50;

% ------------------------------------------------
% Prepare main programm members
% ------------------------------------------------
runCount          = 1;
run               = 1;
generationCount   = 1;
curUnsuccessCount = 1;
curBestQuality    = 1000;
bestQualities     = [];
bestSolutions     = [];
meanQualities     = [];
population        = [];
initialVektor     = [];

% initial candiadte
initialVektor.start1 = 14;
initialVektor.end1   = 20;
initialVektor.start2 = 10;
initialVektor.end2   = 15;
population{1}        = initialVektor;
```

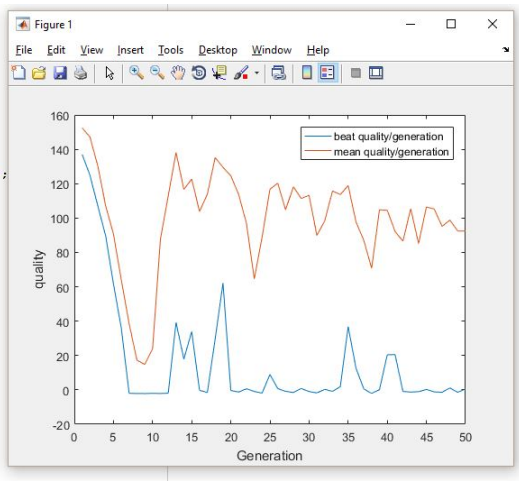

Abbildung 13: $\mu = 5, \lambda = 10, \delta = 10, \delta_{step} = 1, punish = 0.1, mGen = 50, mUnsuc = 50$

2.3.2 Test 2

```
% ------------------------------------------------
% Prepare the algorithm constants
% ------------------------------------------------
cfg                  = [];
cfg.timeSpan         = 100;
cfg.mu               = 5;
cfg.lambda           = 10;
cfg.delta            = 10;
cfg.punish           = 0.1;
cfg.deltaStep        = 1;
cfg.simFile          = 'parametrized_lunar_landing';
cfg.simParams        = simget(cfg.simFile);
cfg.maxValue         = 100;
cfg.maxUnsuccessCount = 100;
cfg.maxGenerations   = 100;

% ------------------------------------------------
% Prepare main programm members
% ------------------------------------------------
runCount          = 1;
run               = 1;
generationCount   = 1;
curUnsuccessCount = 1;
curBestQuality    = 1000;
bestQualities     = [];
bestSolutions     = [];
meanQualities     = [];
population        = [];
initialVektor     = [];

% initial candiadte
initialVektor.start1 = 14;
initialVektor.end1   = 20;
initialVektor.start2 = 10;
initialVektor.end2   = 15;
population{1}        = initialVektor;
```

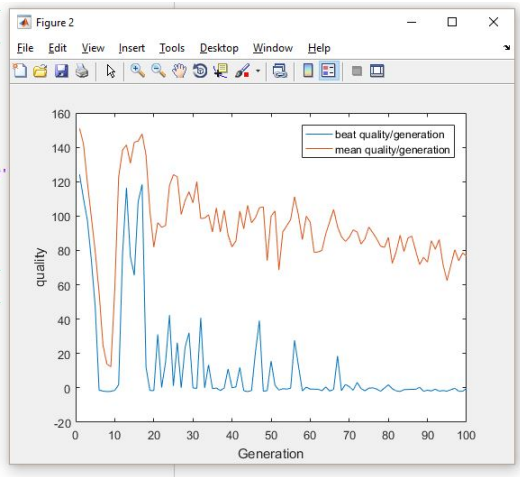

Abbildung 14: $\mu = 5, \lambda = 10, \delta = 10, \delta_{step} = 1, punish = 0.1, mGen = 100, mUnsuc = 100$

2.3.3 Test 3

```
% ------------------------------------------------
% Prepare the algorithm constants
% ------------------------------------------------
cfg                  = [];
cfg.timeSpan         = 100;
cfg.mu               = 5;
cfg.lambda           = 50;
cfg.delta            = 10;
cfg.punish           = 0.1;
cfg.deltaStep        = 1;
cfg.simFile          = 'parametrized_lunar_landing';
cfg.simParams        = simget(cfg.simFile);
cfg.maxValue         = 100;
cfg.maxUnsuccessCount = 50;
cfg.maxGenerations   = 50;

% ------------------------------------------------
% Prepare main programm members
% ------------------------------------------------
runCount          = 1;
run               = 1;
generationCount   = 1;
curUnsuccessCount = 1;
curBestQuality    = 1000;
bestQualities     = [];
bestSolutions     = [];
meanQualities     = [];
population        = [];
initialVektor     = [];

% initial candiadte
initialVektor.start1 = 14;
initialVektor.end1   = 20;
initialVektor.start2 = 10;
initialVektor.end2   = 15;
population{1}        = initialVektor;
```

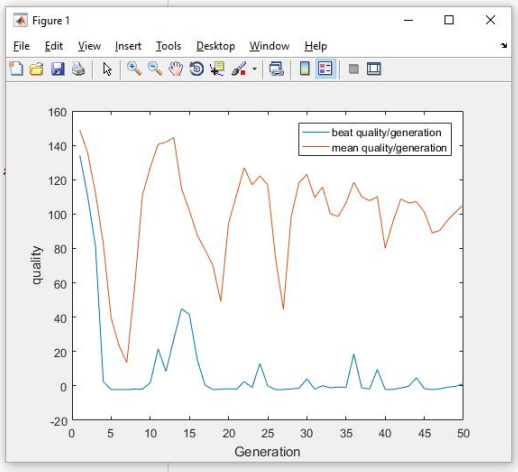

Abbildung 15: $\mu = 5, \lambda = 50, \delta = 10, \delta_{step} = 1, punish = 0.1, mGen = 50, mUnsuc = 50$

2.3.4 Test 4

```
% ------------------------------------------------
% Prepare the algorithm constants
% ------------------------------------------------
cfg                  = [];
cfg.timeSpan         = 100;
cfg.mu               = 5;
cfg.lambda           = 50;
cfg.delta            = 10;
cfg.punish           = 0.1;
cfg.deltaStep        = 1;
cfg.simFile          = 'parametrized_lunar_landing';
cfg.simParams        = simget(cfg.simFile);
cfg.maxValue         = 100;
cfg.maxUnsuccessCount = 100;
cfg.maxGenerations   = 100;

% ------------------------------------------------
% Prepare main programm members
% ------------------------------------------------
runCount          = 1;
run               = 1;
generationCount   = 1;
curUnsuccessCount = 1;
curBestQuality    = 1000;
bestQualities     = [];
bestSolutions     = [];
meanQualities     = [];
population        = [];
initialVektor     = [];

% initial candiadte
initialVektor.start1 = 14;
initialVektor.end1   = 20;
initialVektor.start2 = 10;
initialVektor.end2   = 15;
population{1}        = initialVektor;
```

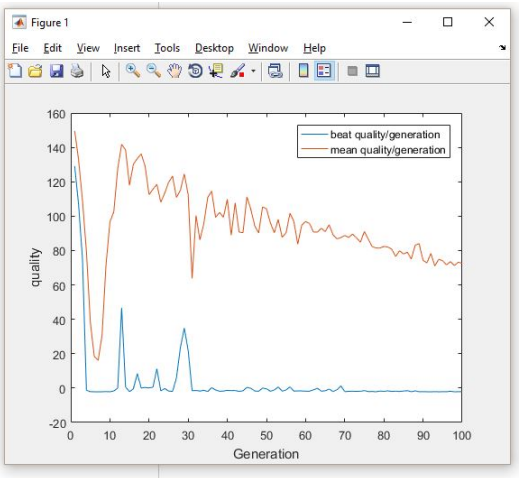

Abbildung 16: $\mu = 5, \lambda = 50, \delta = 10, \delta_{step} = 1, punish = 0.1, mGen = 200, mUnsuc = 100$

2.3.5 Test 5

```
% ------------------------------------------------
% Prepare the algorithm constants
% ------------------------------------------------
cfg                  = [];
cfg.timeSpan         = 100;
cfg.mu               = 5;
cfg.lambda           = 10;
cfg.delta            = 5;
cfg.punish           = 0.1;
cfg.deltaStep        = 0.5;
cfg.simFile          = 'parametrized_lunar_landing';
cfg.simParams        = simget(cfg.simFile);
cfg.maxValue         = 100;
cfg.maxUnsuccessCount = 50;
cfg.maxGenerations   = 50;

% ------------------------------------------------
% Prepare main programm members
% ------------------------------------------------
runCount          = 1;
run               = 1;
generationCount   = 1;
curUnsuccessCount = 1;
curBestQuality    = 1000;
bestQualities     = [];
bestSolutions     = [];
meanQualities     = [];
population        = [];
initialVektor     = [];

% initial candiadte
initialVektor.start1 = 14;
initialVektor.end1   = 20;
initialVektor.start2 = 10;
initialVektor.end2   = 15;
population{1}        = initialVektor;
```

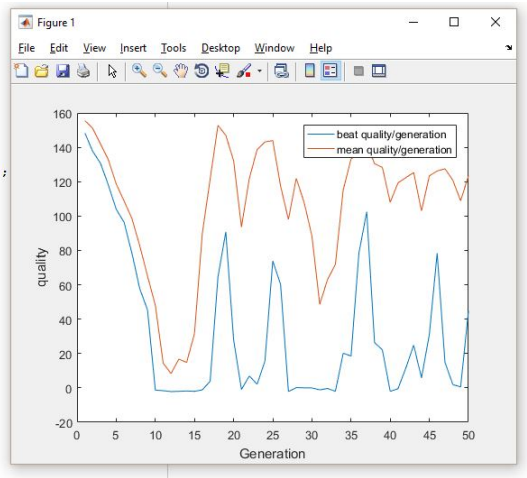

Abbildung 17: $\mu = 5, \lambda = 10, \delta = 5, \delta_{step} = 0.5, punish = 0.1, mGen = 50, mUnsuc = 50$

2.3.6 Test 6

```
% ------------------------------------------------
% Prepare the algorithm constants
% ------------------------------------------------
cfg                  = [];
cfg.timeSpan         = 100;
cfg.mu               = 5;
cfg.lambda           = 10;
cfg.delta            = 5;
cfg.punish           = 0.1;
cfg.deltaStep        = 0.5;
cfg.simFile          = 'parametrized_lunar_landing';
cfg.simParams        = simget(cfg.simFile);
cfg.maxValue         = 100;
cfg.maxUnsuccessCount = 100;
cfg.maxGenerations   = 100;

% ------------------------------------------------
% Prepare main programm members
% ------------------------------------------------
runCount          = 1;
run               = 1;
generationCount   = 1;
curUnsuccessCount = 1;
curBestQuality    = 1000;
bestQualities     = [];
bestSolutions     = [];
meanQualities     = [];
population        = [];
initialVektor     = [];

% initial candiadte
initialVektor.start1 = 14;
initialVektor.end1   = 20;
initialVektor.start2 = 10;
initialVektor.end2   = 15;
population{1}        = initialVektor;
```

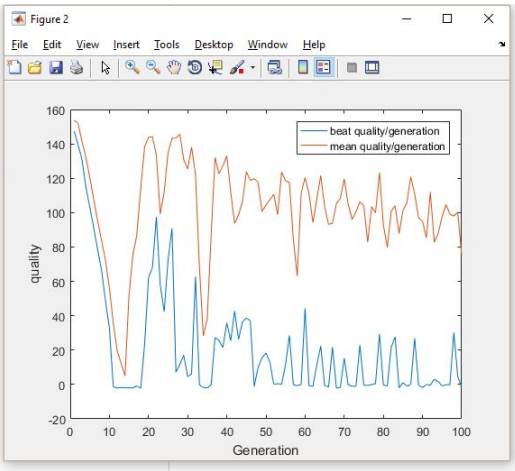

Abbildung 18: $\mu = 5, \lambda = 10, \delta = 5, \delta_{step} = 0.5, punish = 0.1, mGen = 100, mUnsuc = 100$

2.3.7 Test 7

```
% ------------------------------------------------
% Prepare the algorithm constants
% ------------------------------------------------
cfg                  = [];
cfg.timeSpan         = 100;
cfg.mu               = 5;
cfg.lambda           = 50;
cfg.delta            = 5;
cfg.punish           = 0.1;
cfg.deltaStep        = 0.5;
cfg.simFile          = 'parametrized_lunar_landing';
cfg.simParams        = simget(cfg.simFile);
cfg.maxValue         = 100;
cfg.maxUnsuccessCount = 50;
cfg.maxGenerations   = 50;

% ------------------------------------------------
% Prepare main programm members
% ------------------------------------------------
runCount          = 1;
run               = 1;
generationCount   = 1;
curUnsuccessCount = 1;
curBestQuality    = 1000;
bestQualities     = [];
bestSolutions     = [];
meanQualities     = [];
population        = [];
initialVektor     = [];

% initial candiadte
initialVektor.start1 = 14;
initialVektor.end1   = 20;
initialVektor.start2 = 10;
initialVektor.end2   = 15;
population{1}        = initialVektor;
```

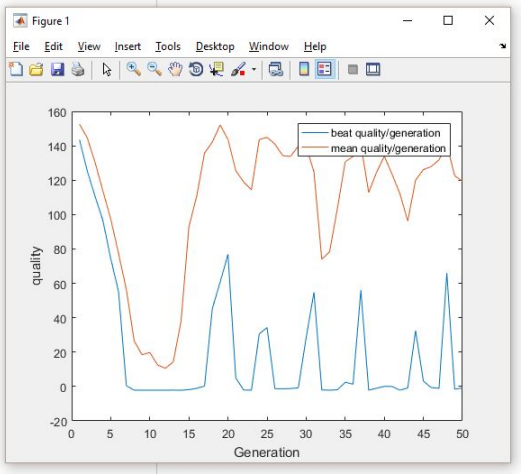

Abbildung 19: $\mu = 5, \lambda = 50, \delta = 5, \delta_{step} = 0.5, punish = 0.1, mGen = 50, mUnsuc = 50$

2.3.8 Test 8

```
% ------------------------------------------------
% Prepare the algorithm constants
% ------------------------------------------------
cfg                  = [];
cfg.timeSpan         = 100;
cfg.mu               = 5;
cfg.lambda           = 50;
cfg.delta            = 5;
cfg.punish           = 0.1;
cfg.deltaStep        = 0.5;
cfg.simFile          = 'parametrized_lunar_landing';
cfg.simParams        = simget(cfg.simFile);
cfg.maxValue         = 100;
cfg.maxUnsuccessCount = 100;
cfg.maxGenerations   = 100;

% ------------------------------------------------
% Prepare main programm members
% ------------------------------------------------
runCount          = 1;
run               = 1;
generationCount   = 1;
curUnsuccessCount = 1;
curBestQuality    = 1000;
bestQualities     = [];
bestSolutions     = [];
meanQualities     = [];
population        = [];
initialVektor     = [];

% initial candiadte
initialVektor.start1 = 14;
initialVektor.end1   = 20;
initialVektor.start2 = 10;
initialVektor.end2   = 15;
population{1}        = initialVektor;
```

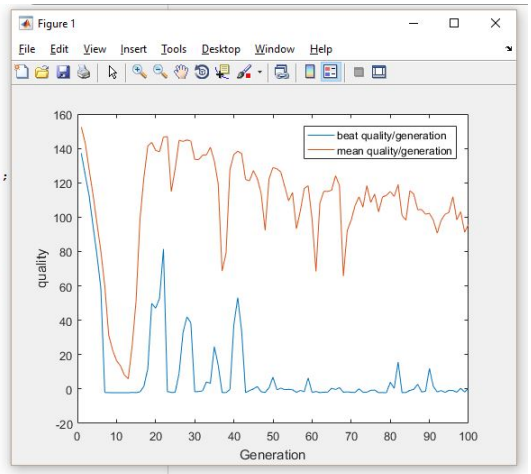

Abbildung 20: $\mu = 5, \lambda = 50, \delta = 5, \delta_{step} = 0.5, punish = 0.1, mGen = 100, mUnsuc = 100$

Die Tests zeigen dass mit einem zu kleinen μ zu viele Lösungskandidaten nicht gefunden werden können. Ebenso muss die Anzahl der Generationen groß genug sein, damit genügend Lösungen durchprobiert werden können.