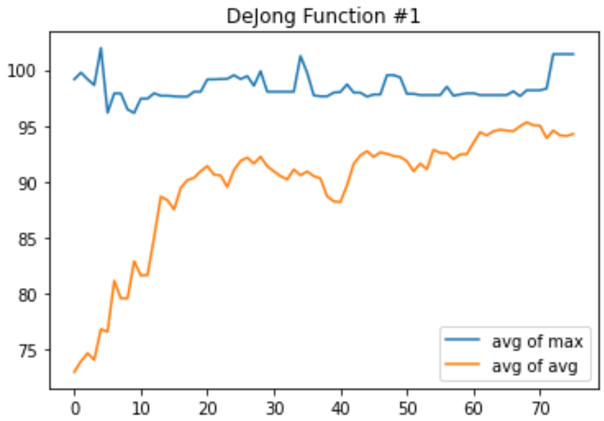
Genetic Algorithms: Assignment 3

David Gabriel

contact@david-gabriel.com

DeJong Function 1

Part 1: Performance



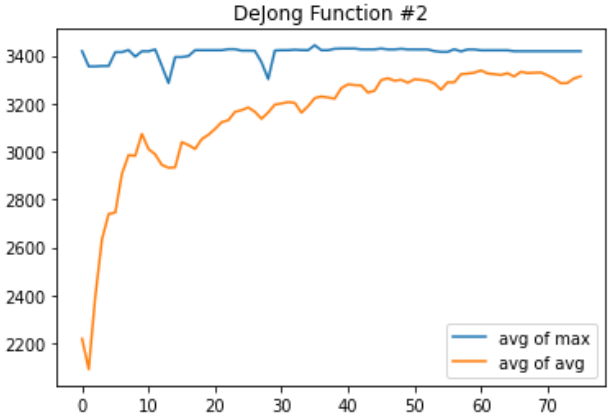
Part 2: Reliability r = 0

Part3: Performance 0%

Part 4: Speed Did not solve within 75 generations

DeJong Function 2

Part 1: Performance



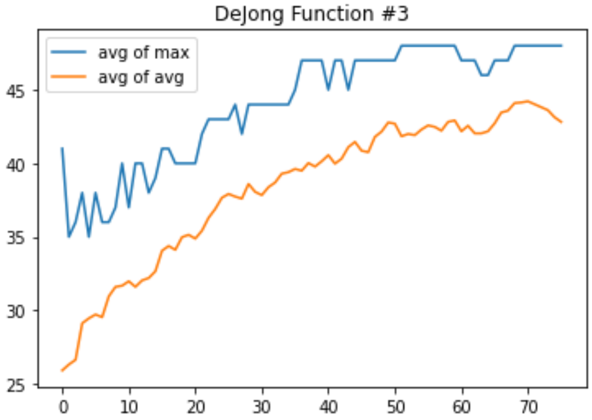
Part 2: Reliability r = 0

Part3: Performance 0%

Part 4: Speed

DeJong Function 3

Part 1: Performance



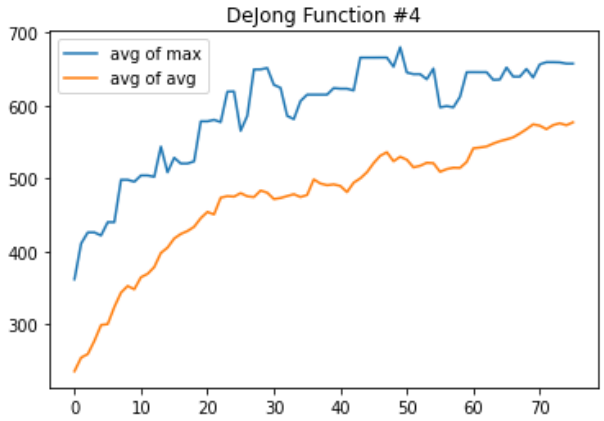
Part 2: Reliability r = 0

Part3: Performance 0%

Part 4: Speed Did not solve within 75 generations

DeJong Function 4

Part 1: Performance



Part 2: Reliability r = 0

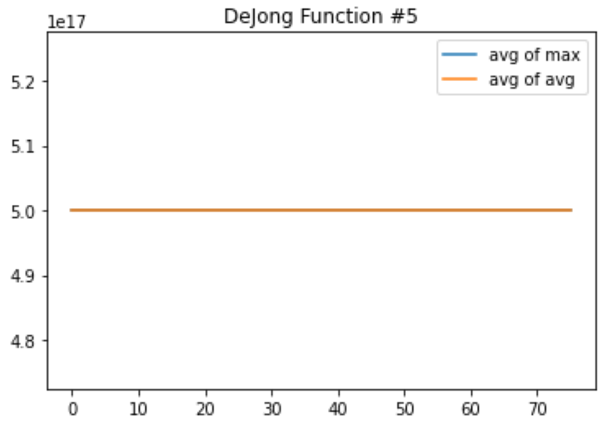
Part3: Performance 0%

Part 4: Speed

DeJong Function 5

Part 1: I am uncertain about this result. I expect issues with my code. I also beleive this to a hard problem. Needs further understanding.

Performance



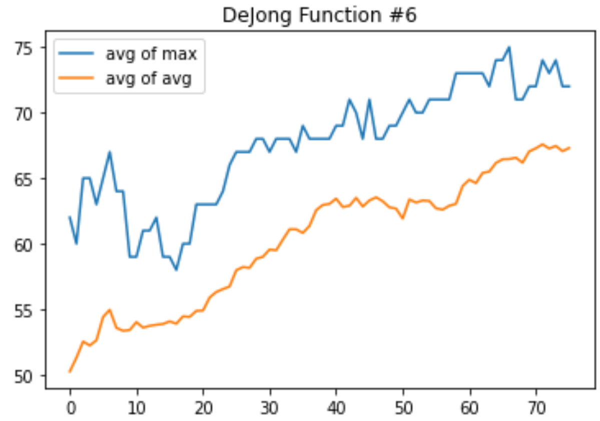
Part 2: Reliability r = ?

Part3: Performance ?

Part 4: Speed Seems to have issues with code

DeJong Function 6

Part 1: Performance



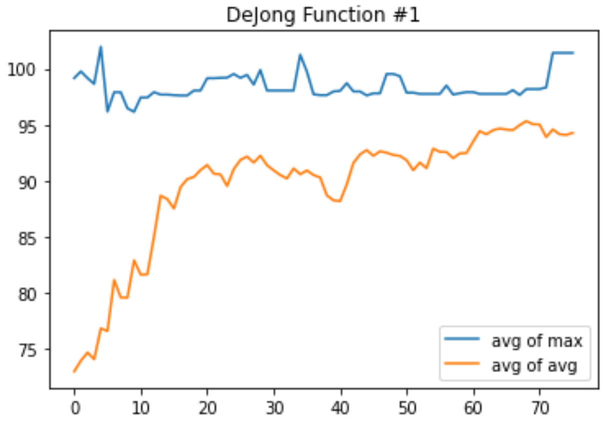
Part 2: Reliability r = 0

Part3: Performance 0%

Part 4: Speed Did not complete within 75 generations

DeJong Function 1

Part 1: Performance



Part 2: Reliability r = 1

Part3: Performance 100%

Part 4: Speed

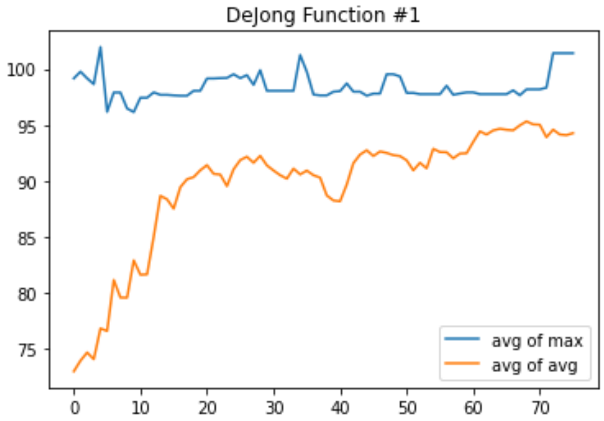
In [31]:
```
import csv
import os
import pandas as pd
import matplotlib.pyplot as plt
```

In [32]:
```
entries = os.listdir('./ga1/results')
results = pd.DataFrame()
for entry in entries:
    new_data = pd.read_csv('./ga1/results/' + entry, delimiter='\s+',header = None)
    results = pd.concat([results, new_data])

#cols = gen, max, avg, min

means = results.groupby(results.index).mean()
#max
#print(means[1])
#avg
#print(means[2])
labels= ['avg of max', 'avg of avg']
fig, ax = plt.subplots()
ax.plot(means.index, means[1])
ax.plot(means.index, means[2])
ax.legend(labels)
ax.set_title('DeJong Function #1')
```

Out[32]:
```
Text(0.5, 1.0, 'DeJong Function #1')
```



In [33]:
```
reliability = results[results.index == 75]
reliability = reliability.groupby(reliability[1]).count()
print(reliability)
reliability = reliability.iloc[0,0]/30
print (reliability)
```

```
         0   2   3   4   5   6   7
1
101.41  30  30  30  30  30  30  30
1.0
```

In [34]:

```
speed =  means[1]
speed
```

Out[34]:

```
0      99.16
1      99.76
2      99.16
3      98.64
4     101.96
       ...  
71     98.33
72    101.41
73    101.41
74    101.41
75    101.41
Name: 1, Length: 76, dtype: float64
```

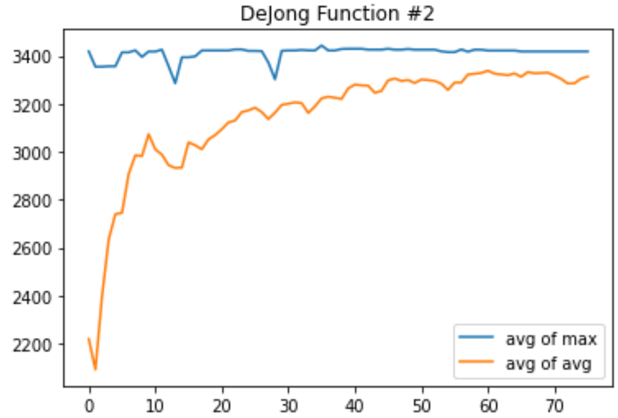
In [35]:

```
del fig, ax
entries = os.listdir('./ga2/results')
results = pd.DataFrame()
for entry in entries:
    new_data = pd.read_csv('./ga2/results/' + entry, delimiter='\s+',header = None)
    results = pd.concat([results, new_data])

#cols = gen, max, avg, min
means = results.groupby(results.index).mean()
#max
print(means[1])
#avg
print(means[2])
labels= ['avg of max', 'avg of avg']
fig, ax = plt.subplots()
ax.plot(means.index, means[1])
ax.plot(means.index, means[2])
ax.legend(labels)
ax.set_title('DeJong Function #2')
```

```
0     3418.71
1     3354.95
2     3354.95
3     3356.82
4     3356.82
       ...
71    3418.44
72    3418.44
73    3418.44
74    3418.44
75    3418.44
Name: 1, Length: 76, dtype: float64
0     2219.63
1     2092.44
2     2398.35
3     2634.77
4     2739.96
       ...
71    3304.26
72    3284.96
73    3285.74
74    3305.15
75    3313.65
Name: 2, Length: 76, dtype: float64
```

```
Out[35]: Text(0.5, 1.0, 'DeJong Function #2')
```



```
In [36]: reliability = results[results.index == 75]
         reliability = reliability.groupby(reliability[1]).count()
         print(reliability)
         reliability = reliability.iloc[0,0]/30
         print (reliability)
```

```
          0   2   3   4   5   6   7
1
3418.44  30  30  30  30  30  30  30
1.0
```

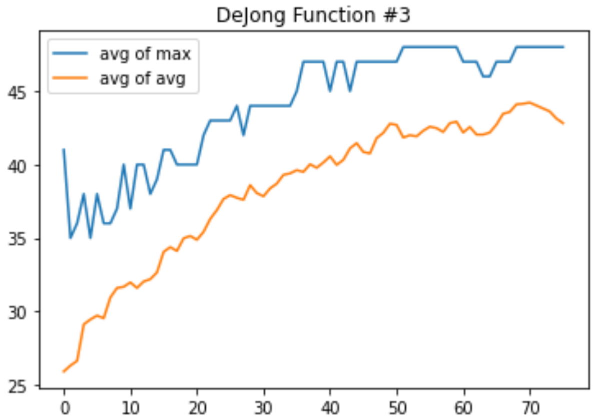
In [37]:

```
del fig, ax
entries = os.listdir('./ga3/results')
results = pd.DataFrame()
for entry in entries:
    new_data = pd.read_csv('./ga3/results/' + entry, delimiter='\s+',header = None)
    results = pd.concat([results, new_data])

#cols = gen, max, avg, min
means = results.groupby(results.index).mean()
#max
print(means[1])
#avg
print(means[2])
labels= ['avg of max', 'avg of avg']
fig, ax = plt.subplots()
ax.plot(means.index, means[1])
ax.plot(means.index, means[2])
ax.legend(labels)
ax.set_title('DeJong Function #3')
```

```
0     41.0
1     35.0
2     36.0
3     38.0
4     35.0
      ...
71    48.0
72    48.0
73    48.0
74    48.0
75    48.0
Name: 1, Length: 76, dtype: float64
0     25.92
1     26.32
2     26.64
3     29.12
4     29.46
      ...
71    44.02
72    43.82
73    43.62
74    43.14
75    42.82
Name: 2, Length: 76, dtype: float64
```

```
Out[37]: Text(0.5, 1.0, 'DeJong Function #3')
```



In [38]:

```
#eliability = results.filter(index = ['75']).groupby(results[1]).count()
reliability = results[results.index == 75]
reliability = reliability.groupby(reliability[1]).count()
print(reliability)
reliability = reliability.iloc[0,0]/30
print (reliability)
s
```

```
       0   2   3   4   5   6   7
1
48.0  30  30  30  30  30  30  30
1.0

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-38-05d3fbdf2c1b> in <module>
      5 reliability = reliability.iloc[0,0]/30
      6 print (reliability)
----> 7 s

NameError: name 's' is not defined
```

In [ ]:

```
del fig, ax
entries = os.listdir('./ga4/results')
results = pd.DataFrame()
for entry in entries:
    new_data = pd.read_csv('./ga4/results/' + entry, delimiter='\s+',header = None)
    results = pd.concat([results, new_data])

#cols = gen, max, avg, min
means = results.groupby(results.index).mean()
#max
print(means[1])
#avg
print(means[2])
labels= ['avg of max', 'avg of avg']
fig, ax = plt.subplots()
ax.plot(means.index, means[1])
ax.plot(means.index, means[2])
ax.legend(labels)
ax.set_title('DeJong Function #4')
```

In [ ]:

```
#eliability = results.filter(index = ['75']).groupby(results[1]).count()
reliability = results[results.index == 75]
reliability = reliability.groupby(reliability[1]).count()
print(reliability)
reliability = reliability.iloc[0,0]/30
print (reliability)
```

In [ ]:

```
del fig, ax
entries = os.listdir('./ga5/results')
results = pd.DataFrame()
for entry in entries:
    new_data = pd.read_csv('./ga5/results/' + entry, delimiter='\s+',header = N
one)
    results = pd.concat([results, new_data])

#cols = gen, max, avg, min
means = results.groupby(results.index).mean()
#max
print(means[1])
#avg
print(means[2])
labels= ['avg of max', 'avg of avg']
fig, ax = plt.subplots()
ax.plot(means.index, means[1])
ax.plot(means.index, means[2])
ax.legend(labels)
ax.set_title('DeJong Function #5')
```

In [ ]:

```
del fig, ax
entries = os.listdir('./ga6/results')
results = pd.DataFrame()
for entry in entries:
    new_data = pd.read_csv('./ga6/results/' + entry, delimiter='\s+',header = N
one)
    results = pd.concat([results, new_data])

#cols = gen, max, avg, min
means = results.groupby(results.index).mean()
#max
print(means[1])
#avg
print(means[2])
labels= ['avg of max', 'avg of avg']
fig, ax = plt.subplots()
ax.plot(means.index, means[1])
ax.plot(means.index, means[2])
ax.legend(labels)
ax.set_title('DeJong Function #6')
```

In [ ]:

```
#eliability = results.filter(index = ['75']).groupby(results[1]).count()
reliability = results[results.index == 75]
reliability = reliability.groupby(reliability[1]).count()
print(reliability)
reliability = reliability.iloc[0,0]/30
print (reliability)
```

Plan:

Write and compile 6 optimizers.

Minimization conversion

Bit space and evaluation

testing

30 trials and stats, change random seed to gen from clock to automate trials

Pt 1:

Bit representation:

1024 bits (ignore -5.12)

5 items, so 50 bits total

Minimzation to maximization

-5.12 : 5.12 ignore -5.12 5.12 ^2 * 5 = 132

fitness = 132 - sum(x_1^2, x_2^2 ... x_5^2)

Answer:

5.12^2 *5 ~132

Pt 2:

Bit representation:

-2.048 : 2.048 ignore -2.048 12 bits per dimension, 2 dimensions 24 bits

Minimzation8 to maximization

Minimize f(x1, x2) = 100(x_1^2 − x_2)^2+ (1−x_1)^2

max (f(x1,x2)): x_1 = 2.048 , x_2 = -2.048 max = 3897.7342268415996

maximize 3898 - 100(x_1^2 − x_2)^2+ (1−x_1)^2

Answer:

~3898

Pt 3:

Bit representation:

Same as pt 1 for x_i

50 bits, 10 bits per 5 inputs

Minimzation to maximization

fitness = 26-f(x)

Answer:

51

Pt 4:

Bit representation:

-1.28 : 1.28 ignore -1.28

256 -> 8 bit

30 dimensions, 8 bits each 240 bits

Minimzation to maximization

max value = 1408

fitness= 1408 - f(x)

Answer:

1408

Pt 5:

Bit representation:

-65.536 : 65.536 ignore -65.536

131072 -> 17 bit

25 dimensions, 17 bits each 425 bits

Minimzation to maximization

max value = 499999999999999937

fitness = 499999999999999937 -f(x)

Answer:

'eval.c' was rewritten the following ways to produce these results C Code Function 1: #include #include /* for pow(x, y) */ #include "type.h" #define n_dim 5 #define bits_per_dim 10 double decode(IPTR pj, int index, int size); double binToDec(int *chrom, int l); double eval(POPULATION *p, IPTR pj) /* Called from gen.c and init.c */ { double val; //double square = 0.0; val = decode(pj, 0, p->lchrom); //square = val * val; return val; } double decode(IPTR pj, int index, int size) { return ((double) binToDec(&

```
(pj->chrom[0]), size)); } double binToDec(int *chrom, int l) { double x[n_dim]; int i; int j = 0; int k = 0; double prod; double sum = 0;
prod = 0.0; for(i = 0; i < l; i++) { j = i % bits_per_dim; if (j == 0) { //x[k] = prod; sum += pow((prod-5.12),2); prod = 0; } prod +=
(chrom[i] == 0 ? 0.0 : pow((double)2.0, (double) j))/100; // printf("prod: %2.2f", prod); } // printf("sum-sqs: %2.2f ", sum); sum = 132
- sum; return sum; } void decToBin(int ad, int *barray, int size) { int i, t; t = ad; for(i = 0; i < size; i++){ barray[i] = t%2; t = t/2; } } C
Code Function 2: #include #include /* for pow(x, y) */ #include "type.h" #define n_dim 5 #define bits_per_dim 10 double
decode(IPTR pj, int index, int size); double binToDec(int *chrom, int l); double eval(POPULATION *p, IPTR pj) /* Called from gen.c
and init.c */ { double val; //double square = 0.0; val = decode(pj, 0, p->lchrom); //square = val * val; return val; } double
decode(IPTR pj, int index, int size) { return ((double) binToDec(&(pj->chrom[0]), size)); } double binToDec(int *chrom, int l) { double
x[n_dim]; int i; int j = 0; int k = 0; double prod; double sum = 0; float x1, x2; prod = 0.0; for(i = 0; i < l; i++) { j = i % bits_per_dim; if (i
== 11 ) { x1 = prod-2.048; prod = 0; } if (i == 23 ) { x2 = prod-2.048; prod = 0; } prod += (chrom[i] == 0 ? 0.0 : pow((double)2.0,
(double) j))/1000; // printf("x1: %2.2f", x1); // printf( "x2: %2.2f\n", x2); } // printf("sum-sqs: %2.2f ", sum); sum = 3898 -
100*pow((pow(x1,2)-x2),2) + pow((1-x1),2); return sum; } void decToBin(int ad, int *barray, int size) { int i, t; t = ad; for(i = 0; i <
size; i++){ barray[i] = t%2; t = t/2; } } C Code: Function 3: #include #include /* for pow(x, y) */ #include "type.h" #define n_dim 5
#define bits_per_dim 10 #define bit_weight 1024 double decode(IPTR pj, int index, int size); double binToDec(int *chrom, int l);
double eval(POPULATION *p, IPTR pj) /* Called from gen.c and init.c */ { double val; //double square = 0.0; val = decode(pj, 0,
p->lchrom); //square = val * val; return val; } double decode(IPTR pj, int index, int size) { return ((double) binToDec(&
(pj->chrom[0]), size)); } double binToDec(int *chrom, int l) { double x[n_dim]; int i; int j = 0; int k = 0; double prod; int skip = 0;
double sum = 0; prod = 0.0; for(i = 0; i < l; i++) { j = i % bits_per_dim; prod += (chrom[i] == 0 ? 0.0 : pow((double)2.0, (double)
j)/100); if (j == bits_per_dim-1 ) { //x[k] = prod; prod = prod -5.12; sum += (int)(prod); prod = 0; } } // printf("sum-sqs: %2.2f ", sum);
sum = 26 - sum; return sum; } void decToBin(int ad, int *barray, int size) { int i, t; t = ad; for(i = 0; i < size; i++){ barray[i] = t%2; t =
t/2; } } C Code: Function 4: #include #include /* for pow(x, y) */ #include "type.h" #include #define n_dim 30 #define bits_per_dim
8 #define bit_weight 256 double decode(IPTR pj, int index, int size); double binToDec(int *chrom, int l); double eval(POPULATION
*p, IPTR pj) /* Called from gen.c and init.c */ { double val; //double square = 0.0; val = decode(pj, 0, p->lchrom); //square = val *
val; return val; } double decode(IPTR pj, int index, int size) { return ((double) binToDec(&(pj->chrom[0]), size)); } double rand_gen()
{ // return a uniformly distributed random value return ( (double)(rand())+1 )/( (double)(RAND_MAX)+1 ); } double normalRandom()
{ // return a normally distributed random value double v1=rand_gen(); double v2=rand_gen(); return
cos(2*3.14*v2)*sqrt(-2.*log(v1)); } double binToDec(int *chrom, int l) { int i; int j = 0; int k = 0; double prod; double sum = 0; prod =
0.0; for(i = 0; i < l; i++) { j = i % bits_per_dim; if (j == 0) { k=floor(i/(bits_per_dim)); sum += k*(pow((prod-1.28),4));// +
normalRandom(); prod = 0; } prod += (chrom[i] == 0 ? 0.0 : pow((double)2.0, (double) j))/100; } // printf("sum-sqs: %2.2f ", sum);
return sum; } void decToBin(int ad, int *barray, int size) { int i, t; t = ad; for(i = 0; i < size; i++){ barray[i] = t%2; t = t/2; } } C Code:
Function 5: #include #include /* for pow(x, y) */ #include "type.h" #include #define n_dim 2 #define bits_per_dim 17 #define
bit_weight 131072 #define bits_per_unit 1000 double decode(IPTR pj, int index, int size); double binToDec(int *chrom, int l);
double eval(POPULATION *p, IPTR pj) /* Called from gen.c and init.c */ { double val; //double square = 0.0; val = decode(pj, 0,
p->lchrom); //square = val * val; return val; } double decode(IPTR pj, int index, int size) { return ((double) binToDec(&
(pj->chrom[0]), size)); } double rand_gen() { // return a uniformly distributed random value return ( (double)(rand()) + 1. )/( (double)
(RAND_MAX) + 1. ); } double normalRandom() { // return a normally distributed random value double v1=rand_gen(); double
v2=rand_gen(); return cos(2*3.14*v2)*sqrt(-2.*log(v1)); } double binToDec(int *chrom, int l) { float x1; float x2; int a1[n_dim] = {-32,-
16,0,16,32,-32,-16,0,16,32,-32,-16,0,16,32,-32,-16,0,16,32,-32,-16,0,16,32}; int a2[n_dim] = {-32,-32,-32,-32,-32,-16,16,-16,16,-
16,0,0,0,0,0,16,16,16,16,16,32,32,32,32,32}; int i; int j = 0; int k = 0; double prod; double sum = 0; prod = 0.0; for(i = 0; i < l; i++) { j
= i % bits_per_dim; if (i == 16 ) { x1 = prod-65.536; prod = 0; } if (i == 33 ) { x2 = prod-65.536; prod = 0; } prod += (chrom[i] == 0 ?
0.0 : pow((double)2.0, (double) j))/1000; } // printf("x1: %2.2f", x1); // printf( "x2: %2.2f\n", x2); prod = 0; // printf("sum-sqs: %2.2f ",
sum); for (i = 1; i < 26; i++) { prod = 1/( i + pow((x1-a1[i]),6) + pow((x2-a2[i]),6) ); sum += prod; sum = 0.002 + sum; } sum =
499999999999999937- sum; return sum; } void decToBin(int ad, int *barray, int size) { int i, t; t = ad; for(i = 0; i < size; i++){
barray[i] = t%2; t = t/2; } } C Code: Function 6: #include #include /* for pow(x, y) */ #include "type.h" #include #define n_dim 100
#define bits_per_dim 1 double decode(IPTR pj, int index, int size); double binToDec(int *chrom, int l); double eval(POPULATION
*p, IPTR pj) /* Called from gen.c and init.c */ { double val; //double square = 0.0; val = decode(pj, 0, p->lchrom); //square = val *
val; return val; } double decode(IPTR pj, int index, int size) { return ((double) binToDec(&(pj->chrom[0]), size)); } double rand_gen()
{ // return a uniformly distributed random value return ( (double)(rand()) + 1. )/( (double)(RAND_MAX) + 1. ); } double
normalRandom() { // return a normally distributed random value double v1=rand_gen(); double v2=rand_gen(); return
cos(2*3.14*v2)*sqrt(-2.*log(v1)); } double binToDec(int *chrom, int l) { int i; double sum = 0; for(i = 0; i < l; i++) { sum += (chrom[i])
== 0 ? 0.0 : 1.0; } return sum; } void decToBin(int ad, int *barray, int size) { int i, t; t = ad; for(i = 0; i < size; i++){ barray[i] = t%2; t =
t/2; } }
```

```
In [39]: x1 = 2.048

         x2 = -2.048
         rosenblat_max = 100* ((x1 ** 2  )-( x2))**2 + ((1 - x1))**2
         rosenblat_max

Out[39]: 3897.7342268415996
```

```
In [40]: #Pt 4:

         sum = 0
         for i in range(30):
             foo = i * 1.28 ** 4 + 8
             sum =  foo + sum

         sum = int(sum) +1
         sum

Out[40]: 1408
```

```
In [41]: # Pt5:
         net = 0
         for i in range(25):
             foo = 0.002 + (1/(  (i+(.001)**6)  + (.001)**6))
             net = foo + net

         net = int(net)
         net

Out[41]: 4999999999999999936
```

```
In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```