In [2]:
```
### ¿¿¿ %matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
```

In [3]:
```
def oracle(i, z, f=lambda x: x):
    if f(i) > f(z):
        return -1
    else:
        return 1
```

Amplitude Amplification with a superposed threshold

We have register 1 $|a\rangle = \sum_i a_i \, |i\rangle$ that will be amplified compared with the threshold value $|b\rangle = \sum_\nu b_\nu \, |\nu\rangle$, in register 2.

Amplitude amplification with a threshold transforms the amplitude a_i of state $|i\rangle$, of the computational basis,

$a_i' = \sum_j \left(\frac{2}{N} - \delta_{ij}\right) \Theta(j, \nu)$, where $\Theta(j, \nu) = \begin{cases} -1 & \text{if } F(j) > F(\nu) \\ 1 & \text{if } F(j) \leq F(\nu) \end{cases}$

Thus for an initial state $|a\rangle \, |b\rangle$ we get the final state

$$|final\rangle = \sum_\nu \sum_i \left\{ b_\nu \sum_j a_j \left(\frac{2}{N} - \delta_{ij}\right) \Theta(j, \nu) \right\} |i\rangle \, |\nu\rangle$$

Uniform in $|a\rangle$, register 1

For a uniform distribution in $|a\rangle$, $a_i = \frac{1}{\sqrt{N}}$, we get the new probability distribution

$$P(i) = \sum_\nu \frac{|b_\nu|^2}{N} \left(2 - \Theta(i, \nu) - 4\frac{t_\nu}{N}\right)^2$$

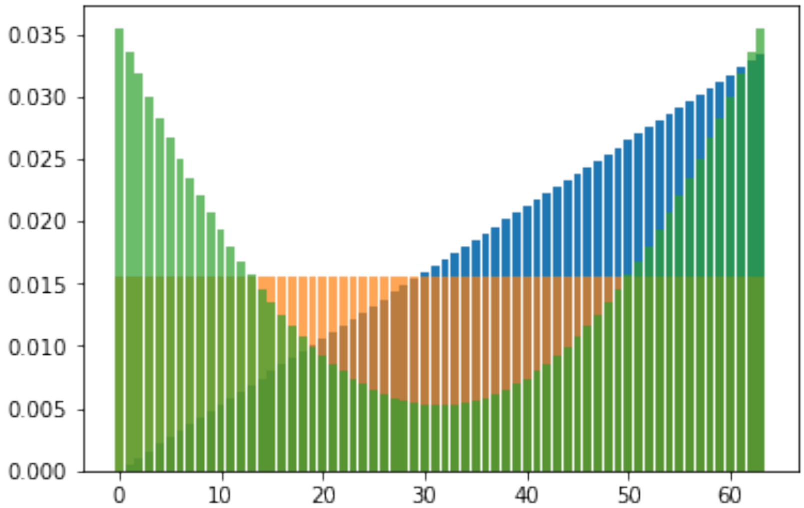
where t_ν is the number of j values that satisfy $\Theta(j, \nu) = -1$, or $F(j) > F(\nu)$.

First we study this case for uniform a_i.

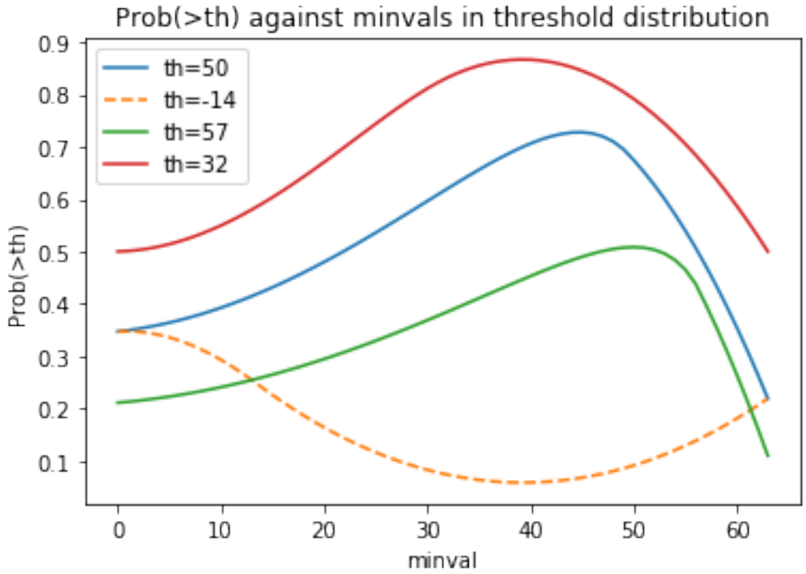
In [4]:
```
# Some distributions

# Step distribution
def step(N, minval, maxval, step):
    bb = np.ones(N)
    if minval>0 and maxval<N:
        bb[minval:maxval] = np.sqrt(step / (maxval - minval))
        bb[:minval] = np.sqrt((1-step) / (N-minval+maxval))
        bb[maxval:] = np.sqrt((1-step) / (N-minval+maxval))
    elif minval == 0 and maxval<N:
        bb[:maxval] = np.sqrt(step / maxval)
        bb[maxval:] = np.sqrt((1-step) / (N+maxval))
    elif maxval == N and minval>0:
        bb[minval:] = np.sqrt(step / (N - minval))
        bb[:minval] = np.sqrt((1-step) / (2*N-minval))
    else:
        bb[:] = 1/np.sqrt(N)
    return bb

# linearly increasing distribution
# nu -> sqrt(nu) !!!
def lininc(N):
    return np.array( [ nu for nu in range(N)]) * np.sqrt( 6 / N / (
N-1) / (2*N-1))

# linearly decreasing distribution
def lindec(N):
    return np.array( [ N-1-nu for nu in range(N)]) * np.sqrt( 6 / N
/ (N-1) / (2*N-1))

# Poisson distribution
def poisson(N, lam):
    return np.sqrt(np.array( [ lam**nu / np.math.factorial(nu) for
nu in range(N)]) * np.exp(-lam))

# Discreticed
def discreticed(N, iarr):
    bb = np.zeros(N)
    bb[iarr] = 1/np.sqrt(len(iarr))
    return bb
```

In [4]:
```
N = 64
ii = np.arange(N)
pp = np.zeros(N)
minval = 0
bb = step(N, minval=minval, maxval=N, step=1)
```

```
minval: 0
Prob >49: 0.3469
Prob <14: 0.3469
Prob >57: 0.2108
```



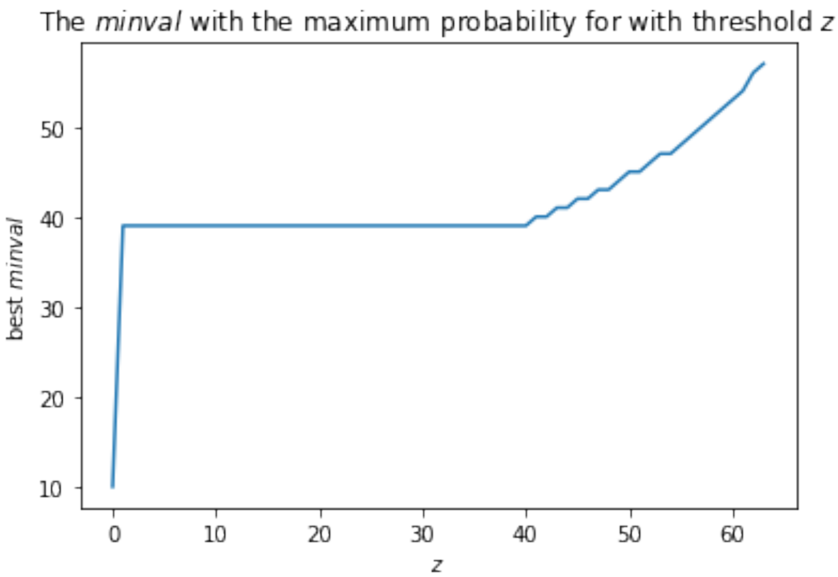
In [5]:
```
top = []
top_th = 50
bot = []
bot_th = 14

excel = []
excel_th = 57
good = []
good_th = 32

for mv in range(0, N):
    pp = np.zeros(N)
    bb = step(N, minval=mv, maxval=N, step=1)

    function = lambda x: x

    for i in ii:
        for nu  in range(N):
            pp[i] += bb[nu]**2 / N * (2 - oracle(i, nu, f=function)
- 4 *(N-1-nu)/N )**2

    top.append(sum(pp[top_th:]))
    bot.append(sum(pp[:bot_th]))

    excel.append(sum(pp[excel_th:]))
    good.append(sum(pp[good_th:]))

plt.plot(top, label='th={}'.format(top_th))
plt.plot(bot,'--', label='th=-{}'.format(bot_th))
plt.plot(excel, label='th={}'.format(excel_th))
plt.plot(good, label='th={}'.format(good_th))

plt.legend()
plt.title('Prob(>th) against minvals in threshold distribution')
plt.ylabel('Prob(>th)')
plt.xlabel('minval')

plt.show()
```

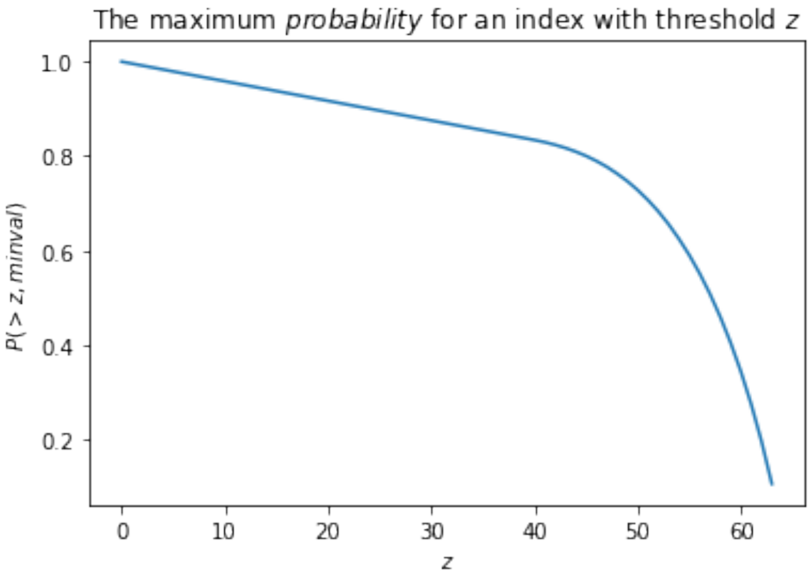


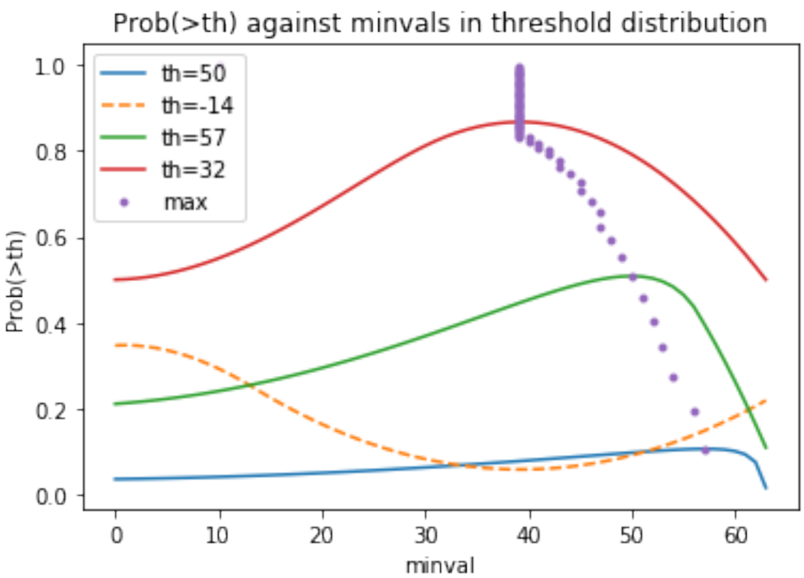
In [6]:
```
# How does the position of the maximum vary?
sup = []
isup = []

for thres in range(0, N):
    top = []
    for mv in range(0, N):
        pp = np.zeros(N)
        bb = step(N, minval=mv, maxval=N, step=1)

        function = lambda x: x

        for i in ii:
            for nu  in range(N):
                pp[i] += bb[nu]**2 / N * (2 - oracle(i, nu, f=funct
ion) - 4 *(N-1-nu)/N )**2

        top.append(sum(pp[thres:]))

    sup.append(max(top))
    for i, s in enumerate(top):
        if s == sup[-1]:
            isup.append(i)
            break
```

In [7]:

```
plt.plot(isup)
plt.title('The $minval$ with the maximum probability for with thres
hold $z$')
plt.ylabel('best $minval$')
plt.xlabel('$z$')
plt.show()

plt.plot(sup)
plt.title('The maximum $probability$ for an index with threshold $z
$')
plt.ylabel('$P(>z, minval)$')
plt.xlabel('$z$')
plt.show()

plt.plot(top, label='th={}'.format(top_th))
plt.plot(bot,'--', label='th=-{}'.format(bot_th))
plt.plot(excel, label='th={}'.format(excel_th))
plt.plot(good, label='th={}'.format(good_th))

plt.plot(isup, sup, '.', label='max')

plt.legend()
plt.title('Prob(>th) against minvals in threshold distribution')
plt.ylabel('Prob(>th)')
plt.xlabel('minval')

plt.show()
```


The $minval$ with the maximum probability for with threshold z
best $minval$
z
10
20
30
40
50
0
10
20
30
40
50
60


The maximum $probability$ for an index with threshold z
$P(> z, minval)$
z
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40
50
60


Prob(>th) against minvals in threshold distribution
Prob(>th)
minval
th=50
th=-14
th=57
th=32
max
0.0
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40
50
60

NON-uniform in $|a\rangle$, register 1

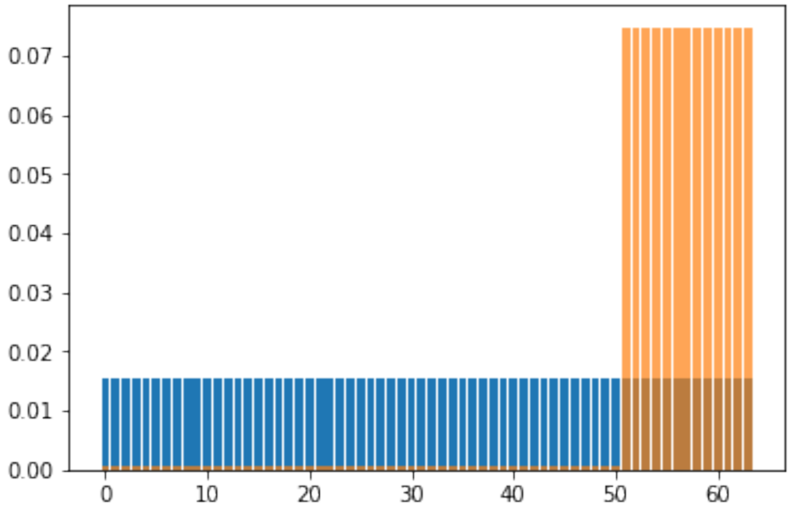
For a non-uniform distribution in $|a\rangle$, we get the new probability distribution

$$P(i) = \sum_{\nu} \left| b_{\nu} \sum_{j} a_j \left(\frac{2}{N} - \delta_{ij} \right) \Theta(j, \nu) \right|^2$$

In [6]:
```
def prob(i, a, b, function=lambda x: x):
    # assumes the given amplitudes are real
    N = len(a)

    p = 0
    for nu in range(N):

        pnu = 0
        for j in range(N):
            pnu += a[j] * oracle(j, nu, function)
        pnu *= 2/N
        pnu -= a[i] * oracle(i, nu, function)
        pnu *= b[nu]

        p += pnu**2
    return p
```

In [9]:
```
N = 64
ii = np.arange(N)
pp = np.zeros(N)
mina, maxa = 0, N
minb, maxb = 45, 55
aa = step(N, minval=mina, maxval=maxa, step=1)
bb = discreticed(N, [50])#step(N, minval=minb, maxval=maxb, step=1)

function = lambda x: x

for i in ii:
    pp[i] += prob(i, aa, bb, function=function)

print(' {:10s} | {:10s} | {:10s} | {:10s} | {:10s}'.format('a range
', 'b range', 'bad (<14)', 'good (>49)', 'excel (>57)'))
print(' {:10s} | {:10s} | {:10s} | {:10s} | {:10s}'.format('-'*10,
'-'*10, '-'*10, '-'*10, '-'*10))
print(' [{:3}, {:3}) | [{:3}, {:3}) | {:10f} | {:10f} | {:10f}'.for
mat(mina, maxa, minb, maxb , sum(pp[:14]), sum(pp[50:]), sum(pp[57:
])))

plt.bar(ii, aa*aa)
#plt.bar(ii, bb*bb, alpha=0.7)
plt.bar(ii, pp, alpha=0.7)
plt.show()
```

```
 a range    | b range    | bad (<14)  | good (>49) | excel (>57)
 ---------- | ---------- | ---------- | ---------- | ----------
 [  0,  64) | [ 45,  55) |   0.007690 |   0.972534 |   0.523376
```

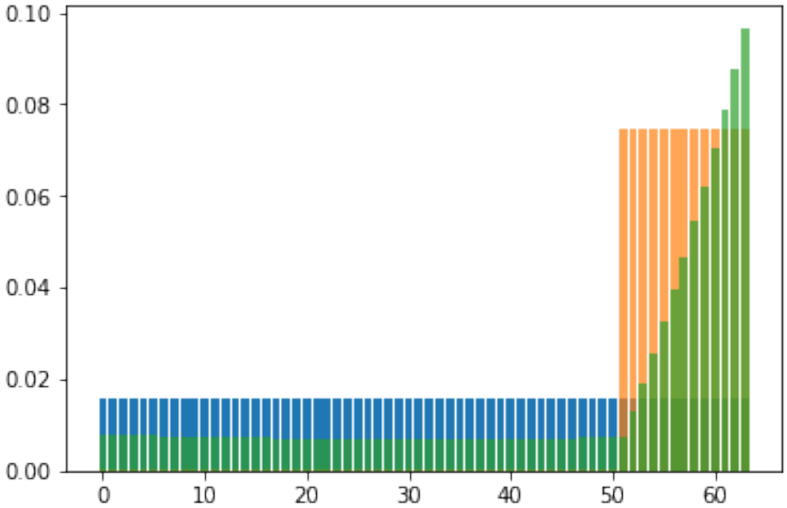


In [10]:
```
N = 64
ii = np.arange(N)

print('| {:10s} | {:10s} | {:10s} | {:10s} | {:10s}|'.format('a ran
ge', 'b range', 'bad (<14)', 'good (>49)', 'excel (>57)'))

arange_arr = [[32, N], [50, N], [32, N], [32, N], [0, 32], [32, N],
[45, 55]]
brange_arr = [[50, N], [32, N], [32, N], [0, 32], [32, N], [0, 50],
[45, 55]]
for arange, brange in zip(arange_arr, brange_arr):
    pp = np.zeros(N)
    mina, maxa = arange
    minb, maxb = brange
    aa = step(N, minval=mina, maxval=maxa, step=1)
    bb = step(N, minval=minb, maxval=maxb, step=1)

    function = lambda x: x

    for i in ii:
        pp[i] += prob(i, aa, bb, function=function)

    print('| {:10s} | {:10s} | {:10s} | {:10s} | {:10s} |'.format('
-'*10, '-'*10, '-'*10, '-'*10, '-'*10))
    print('| [{:3}, {:3}) | [{:3}, {:3}) | {:10f} | {:10f} | {:10f}
|'.format(mina, maxa, minb, maxb , sum(pp[:14]), sum(pp[50:]), sum(
pp[57:])))
```

```
| a range    | b range    | bad (<14)  | good (>49) | excel (>57)|
| ---------- | ---------- | ---------- | ---------- | ---------- |
| [ 32,  64) | [ 50,  64) |   0.182007 |   0.455444 |   0.350769 |
| ---------- | ---------- | ---------- | ---------- | ---------- |
| [ 50,  64) | [ 32,  64) |   0.135864 |   0.514771 |   0.257385 |
| ---------- | ---------- | ---------- | ---------- | ---------- |
| [ 32,  64) | [ 32,  64) |   0.146118 |   0.311890 |   0.209778 |
| ---------- | ---------- | ---------- | ---------- | ---------- |
| [ 32,  64) | [  0,  32) |   0.437500 |   0.000000 |   0.000000 |
| ---------- | ---------- | ---------- | ---------- | ---------- |
| [  0,  32) | [ 32,  64) |   0.000000 |   0.437500 |   0.218750 |
| ---------- | ---------- | ---------- | ---------- | ---------- |
| [ 32,  64) | [  0,  50) |   0.322554 |   0.072085 |   0.036042 |
| ---------- | ---------- | ---------- | ---------- | ---------- |
| [ 45,  55) | [ 45,  55) |   0.046484 |   0.440234 |   0.023242 |
```

In [11]:
```
N = 64

z = 50 # Has to be higher than 31 (N/2 - 1) to work
function = lambda x: x

ii = np.arange(N)
pp = np.zeros(N)

aa = step(N, minval=0, maxval=N, step=1)

bb = np.zeros(N)
for nu in range(N):
    for mu in range(N):
        bb[nu] += oracle(mu, z, f=function)
    bb[nu] *= 2/N
    bb[nu] -= oracle(nu, z, f=function)
    bb[nu] /= np.sqrt(N)

for i in ii:
    pp[i] += prob(i, aa, bb, function=function)

print('Threshold (if any):    ', z)
print('| {:10s} | {:10s} | {:10s}'.format('bad (<14)', 'good (>49)'
, 'excel (>57)'))
print('| {:10s} | {:10s} | {:10s}'.format('-'*10, '-'*10, '-'*10))
print('| {:10f} | {:10f} | {:10f}'.format( sum(pp[:14]), sum(pp[50:
]), sum(pp[57:])))

plt.bar(ii, aa*aa)
plt.bar(ii, bb*bb, alpha=0.7)
plt.bar(ii, pp, alpha=0.7)
plt.show()
```

```
Threshold (if any):    50
| bad (<14)  | good (>49) | excel (>57)
| ---------- | ---------- | ----------
|   0.106184 |   0.641371 |   0.496820
```



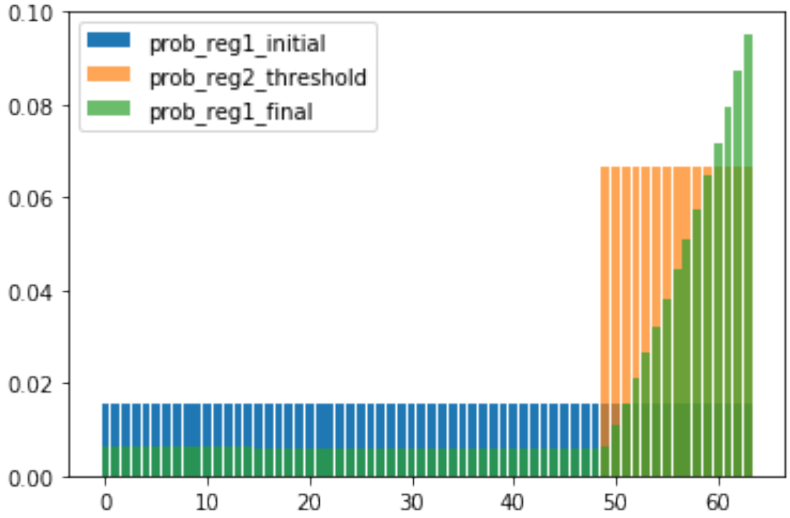
Testing different fitness functions

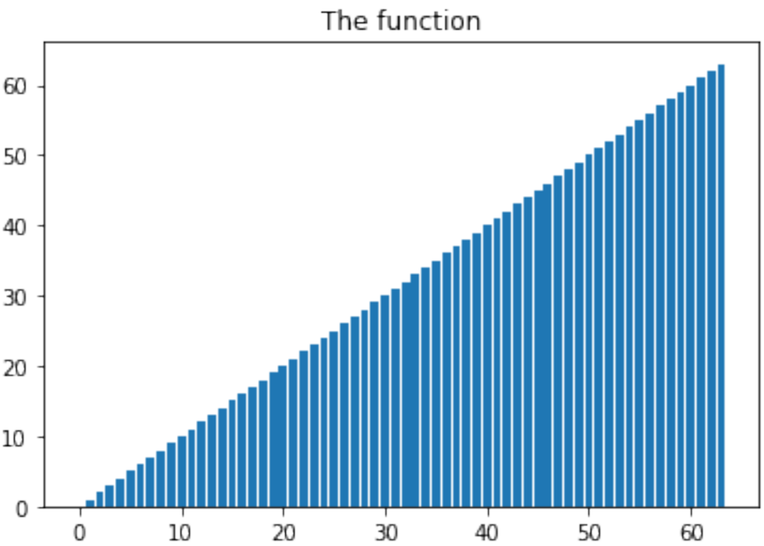
There is a range where appliying amplitude amplification amplifies the amplitude of a desired set, but depending on the state used and the number of solutions the result may be very different.

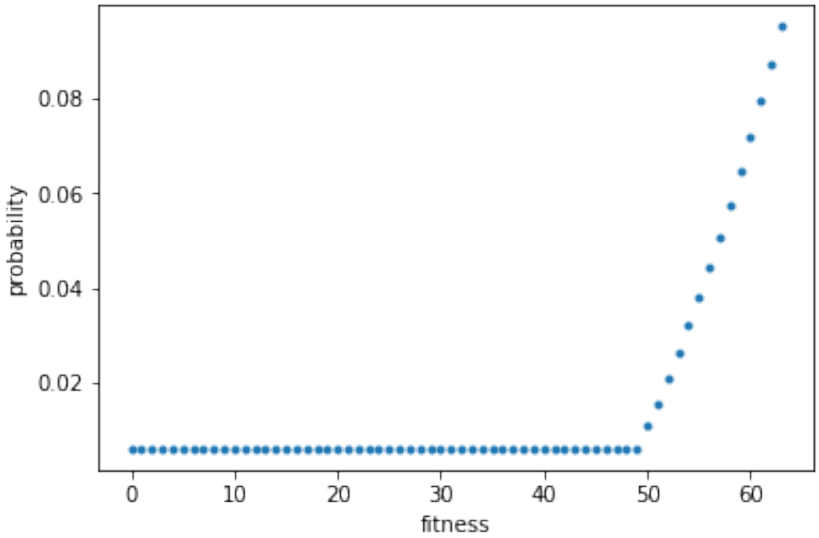
In [5]:
```
def test_function(N, aa, funcname, function, z):
    ii = np.arange(N)
    bb = np.zeros(N)
    for nu in range(N):
        for mu in range(N):
            bb[nu] += oracle(mu, z, f=function)
        bb[nu] *= 2/N
        bb[nu] -= oracle(nu, z, f=function)
        bb[nu] /= np.sqrt(N)

    pp = np.zeros(N)
    for i in ii:
        pp[i] += prob(i, aa, bb, function=function)

    print('| {:12s} | {:12s} | {:12s} | {:12s} | {:12s}|'.format('f
unction', 'z & tz/N', 'bad (f<%20)', 'good (f>%80)', 'excel (f>%90)
'))
    print('| {:12s} | {:12s} | {:12s} | {:12s} | {:12s} |'.format('
-'*12, '-'*12, '-'*12, '-'*12, '-'*12))
    fp = [(function(i), pp[i]) for i in ii]
    fp.sort()
    bad = sum(fp[i][1] for i in range(N//5))
    good = sum(fp[i][1] for i in range(N-N//5, N))
    excel = sum(fp[i][1] for i in range(N-N//10, N))
    tz = sum(1 for i in range(N) if function(i)>function(z))
    print('| {:>12s} | {:3} & {:5.4f} | {:12f} | {:12f} | {:12f} |'
.format(funcname, z, tz/N, bad, good, excel))

    plt.bar(ii, aa*aa, label='prob_reg1_initial')
    plt.bar(ii, bb*bb, alpha=0.7, label='prob_reg2_threshold')
    plt.bar(ii, pp, alpha=0.7, label='prob_reg1_final')
    plt.legend()
    plt.show()

    plt.title('The function')
    plt.bar(ii, function(ii))
    plt.show()

    plt.plot(function(ii), pp, '.')
    plt.ylabel('probability')
    plt.xlabel('fitness')
    plt.show()
```

In [13]:
```
N = 64
aa = step(N, minval=0, maxval=N, step=1)
test_function(N, aa, 'x', lambda x: x, 48)
```

```
| function      | z & tz/N     | bad (f<%20)  | good (f>%80) | exce
l (f>%90)|
| ------------ | ------------ | ------------ | ------------ | ----
-------- |
|            x |  48 & 0.2344 |     0.073941 |     0.667966 |
0.455493 |
```

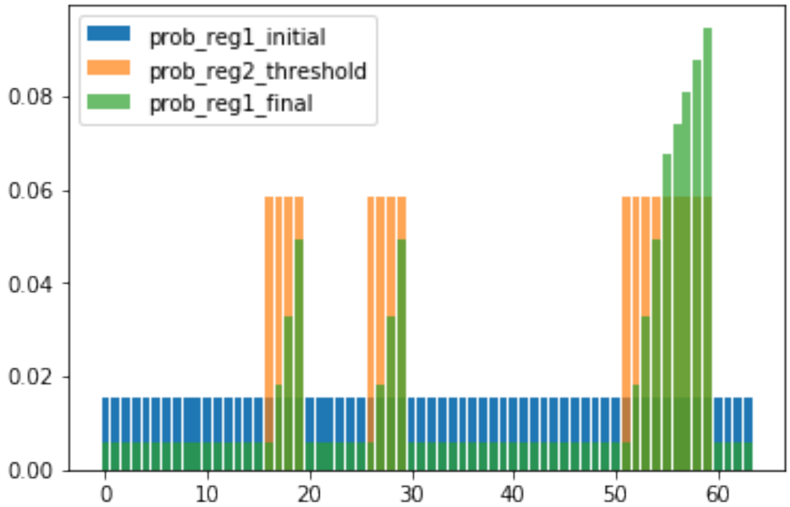


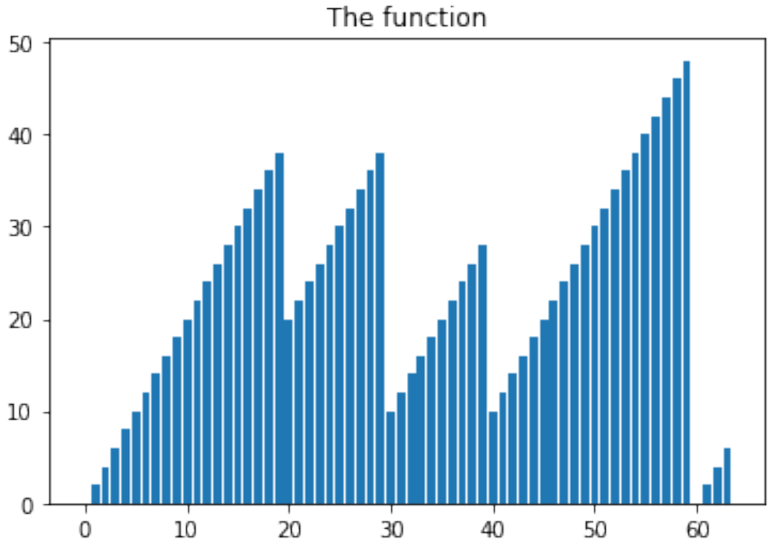


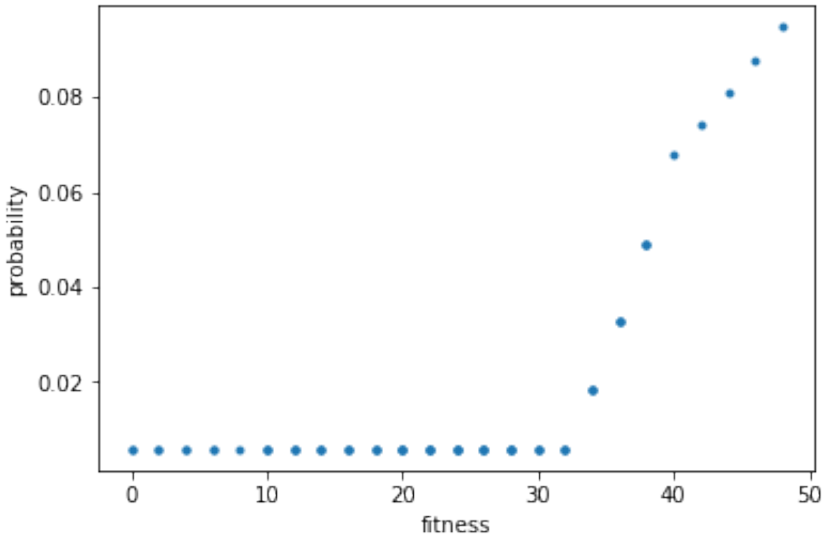


In [14]:
```
N = 64
aa = step(N, minval=0, maxval=N, step=1)
test_function(N, aa, 'slopes', lambda x: x % 20 + x % 30, 50)
```

```
| function     | z & tz/N     | bad (f<%20)  | good (f>%80) | exce
l (f>%90)|
| ------------ | ------------ | ------------ | ------------ | ----
-------- |
|       slopes |  50 & 0.2656 |     0.071228 |     0.669150 |
0.454405 |
```

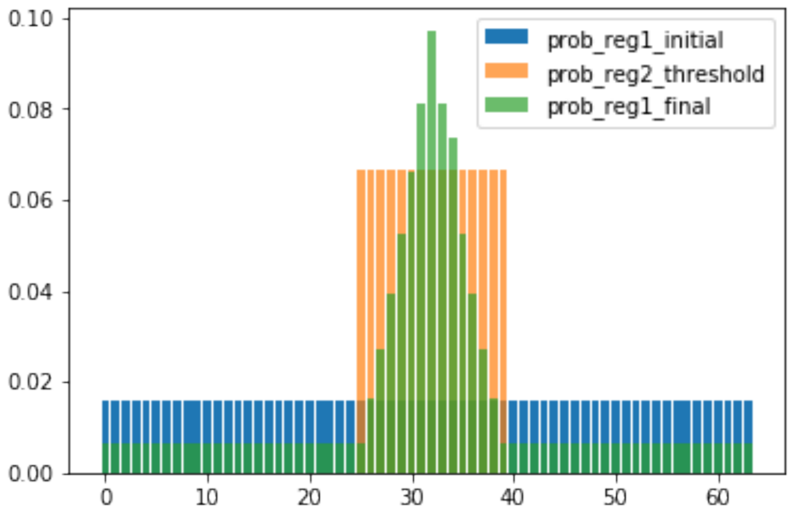


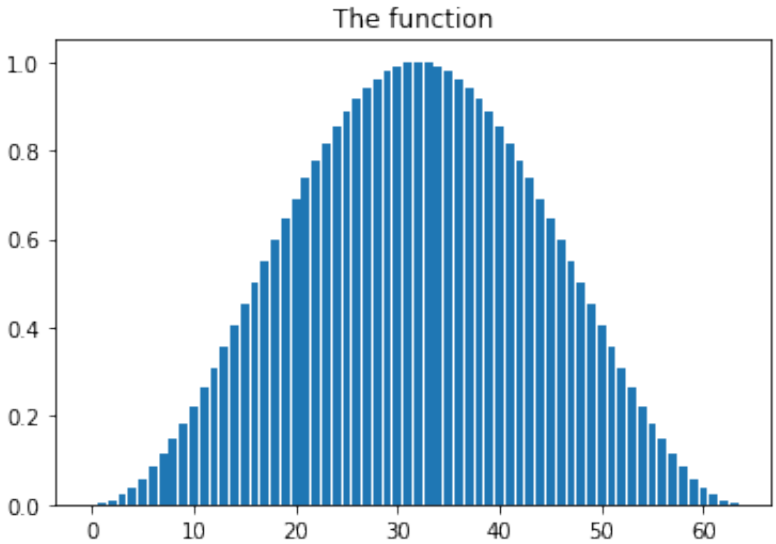


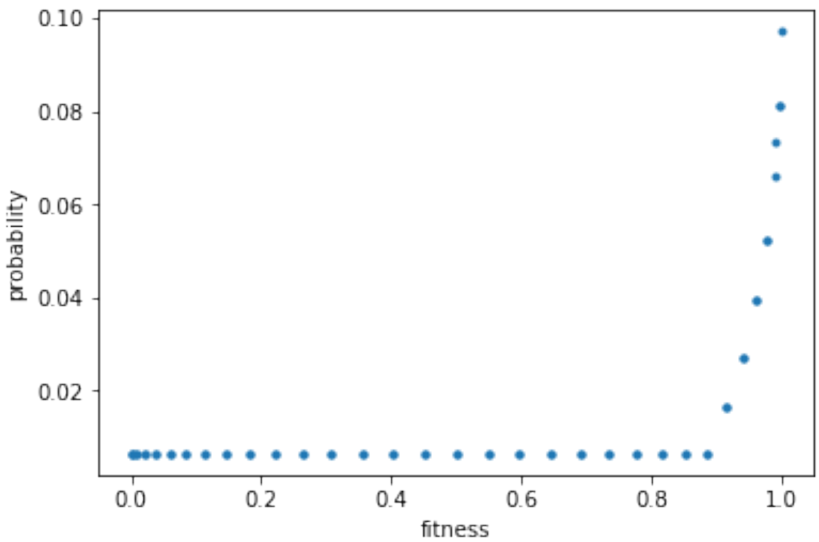


In [15]:
```
N = 64
aa = step(N, minval=0, maxval=N, step=1)
test_function(N, aa, 'sin(pi*x/N)^2', lambda x: np.sin(np.pi*x/N)**
2, 40)
```

```
| function     | z & tz/N     | bad (f<%20)  | good (f>%80) | exce
l (f>%90)|
| ------------ | ------------ | ------------ | ------------ | ----
-------- |
| sin(pi*x/N)^2 |  40 & 0.2344 |     0.078509 |     0.652558 |
0.451035 |
```

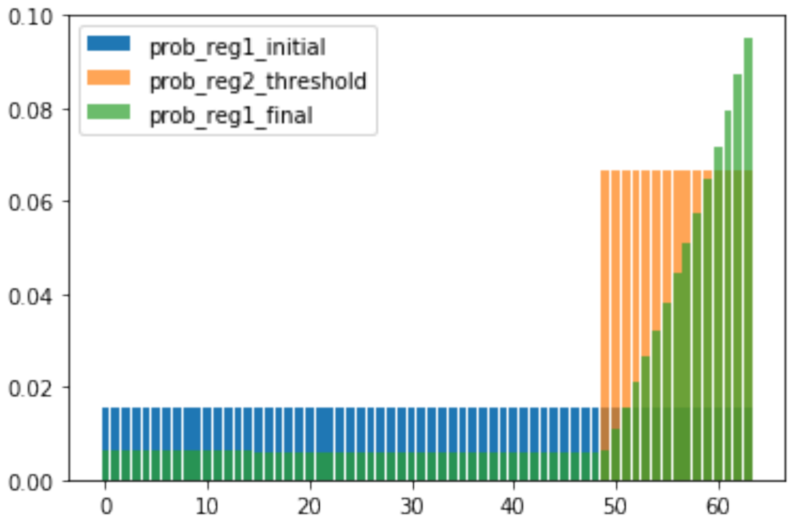


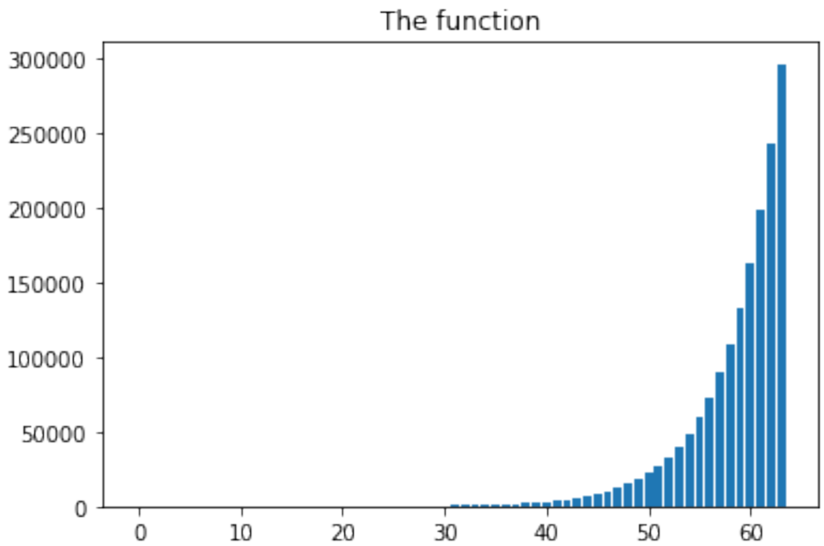


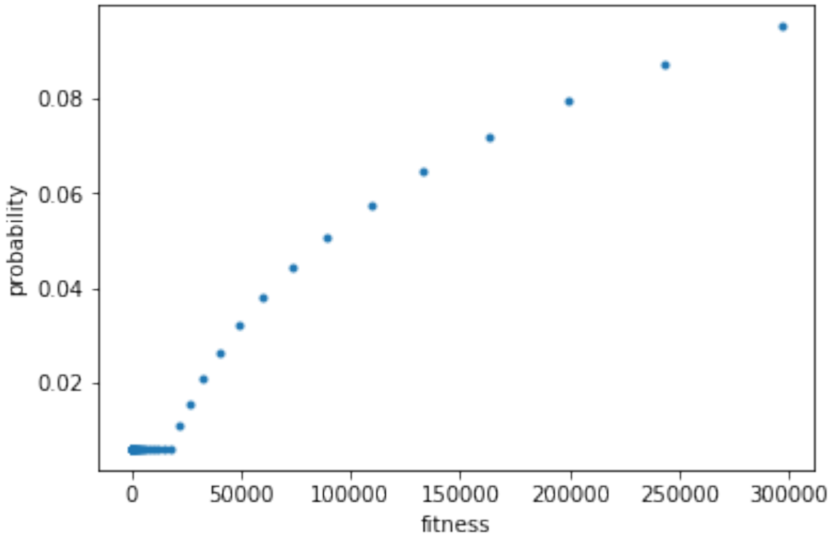


In [16]:
```
N = 64
aa = step(N, minval=0, maxval=N, step=1)
test_function(N, aa, 'exp(x/5)', lambda x: np.exp(x/5), 48)
```

function	z & tz/N	bad (f<%20)	good (f>%80)	excel (f>%90)
exp(x/5)	48 & 0.2344	0.073941	0.667966	0.455493



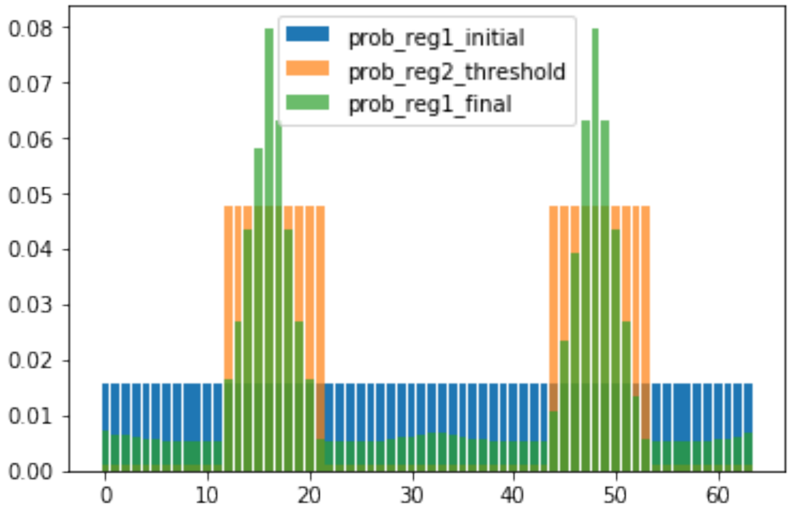


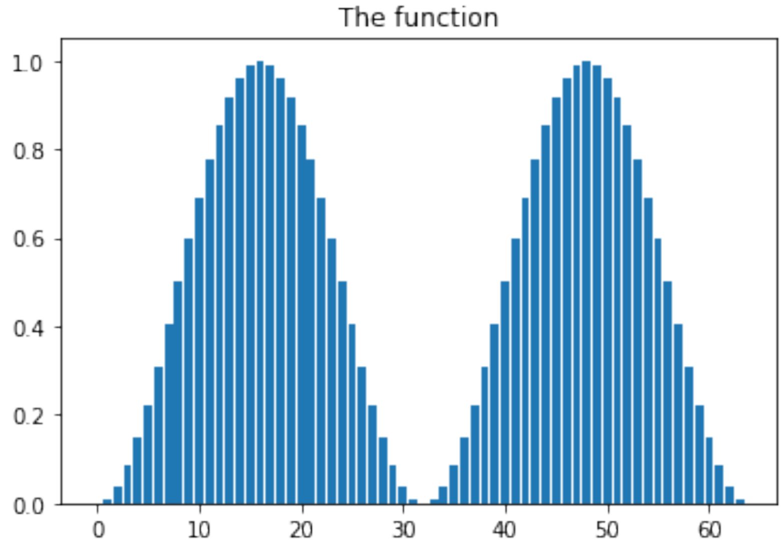


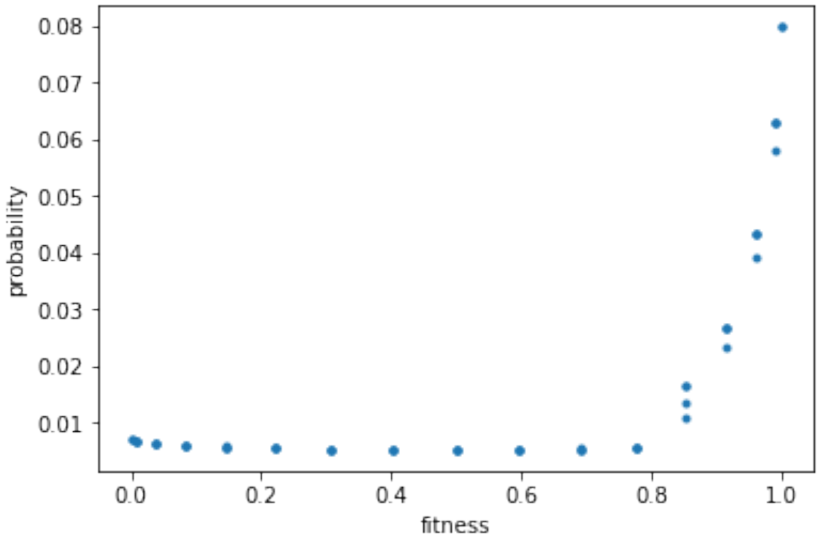
In [17]:

```
N = 64
aa = step(N, minval=0, maxval=N, step=1)
test_function(N, aa,'sin(2*pi*x/N)^2', lambda x: np.sin(2*np.pi*x/N
)**2, 11)
```

```
| function      | z & tz/N      | bad (f<%20)   | good (f>%80) | exce
l (f>%90)|
| ------------ | ------------ | ------------ | ------------ | ----
-------- |
| sin(2*pi*x/N)^2 |  11 & 0.3125 |     0.078030 |     0.630399 |
0.407164 |
```



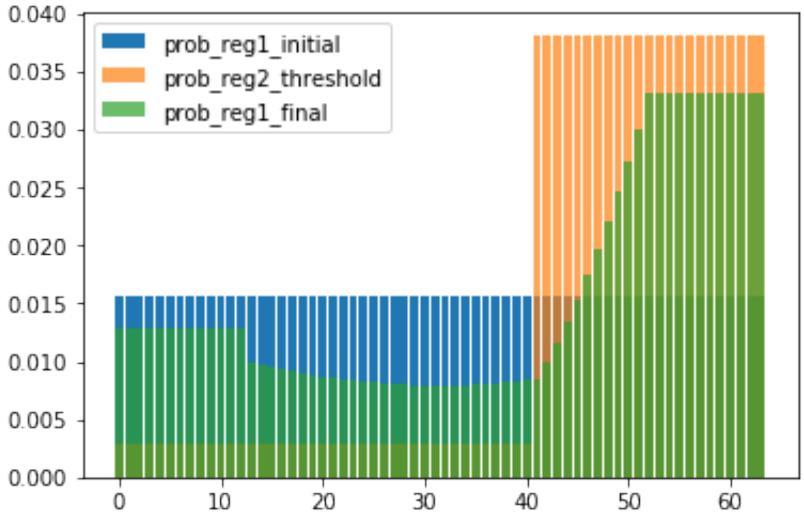


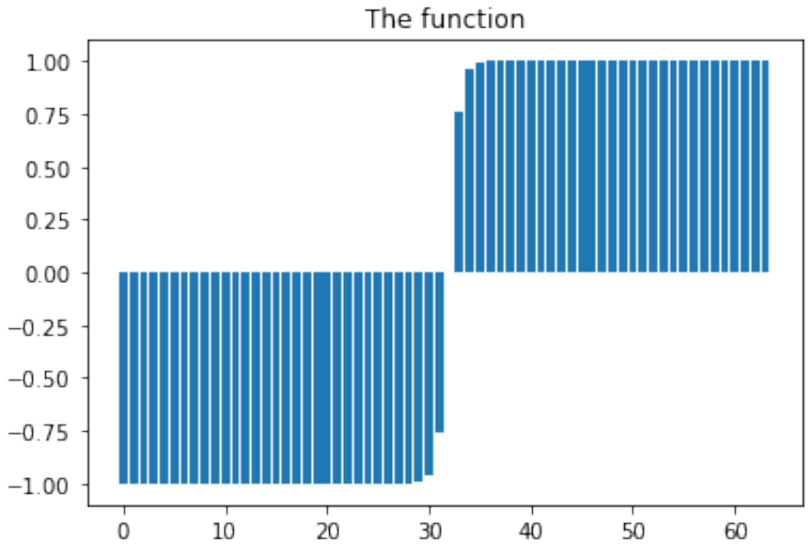


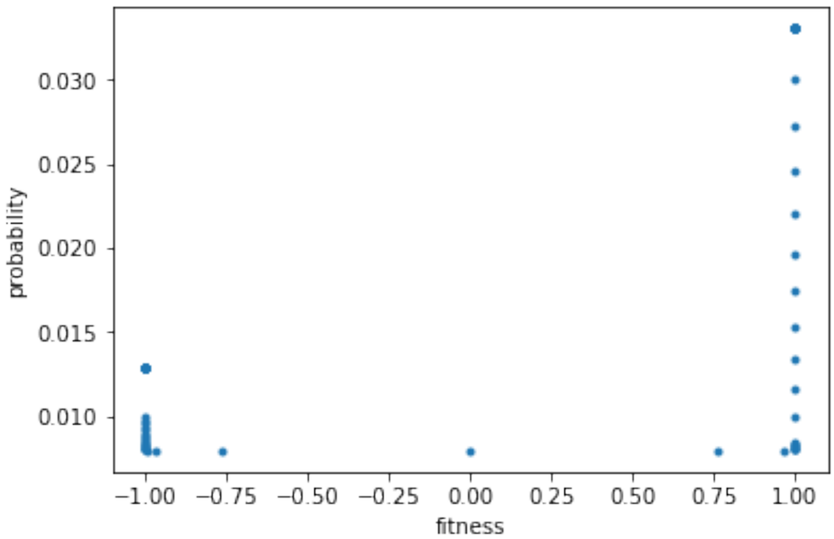
In [18]:

```
N = 64
aa = step(N, minval=0, maxval=N, step=1)
test_function(N, aa,'tanh((x-N/2))', lambda x: np.tanh((x-N/2)), 40
)
```

function	z & tz/N	bad (f<%20)	good (f>%80)	excel (f>%90)
tanh((x-N/2))	40 & 0.3594	0.153872	0.396624	0.198312

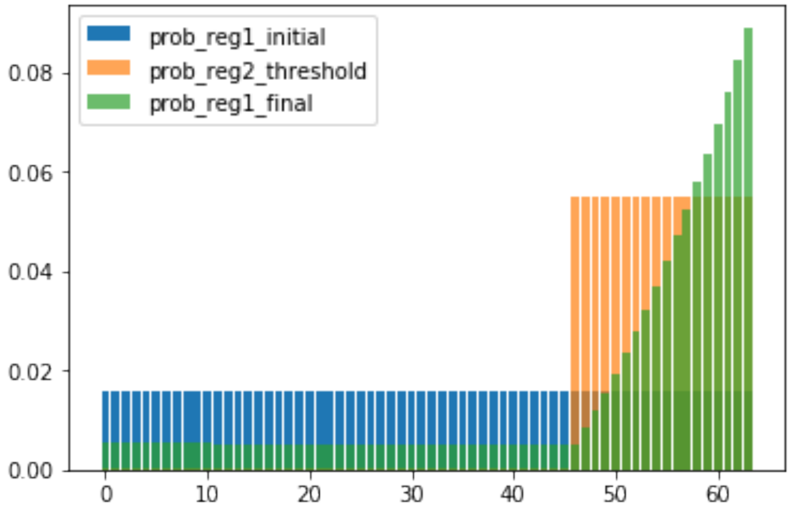


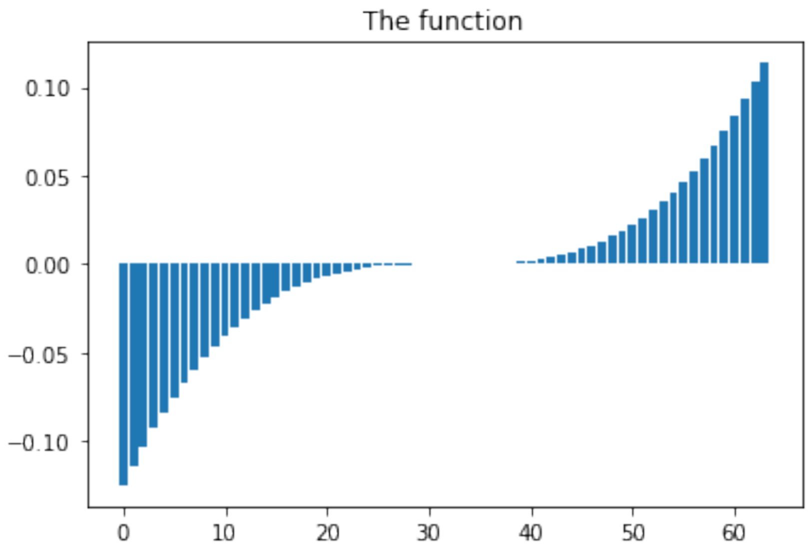


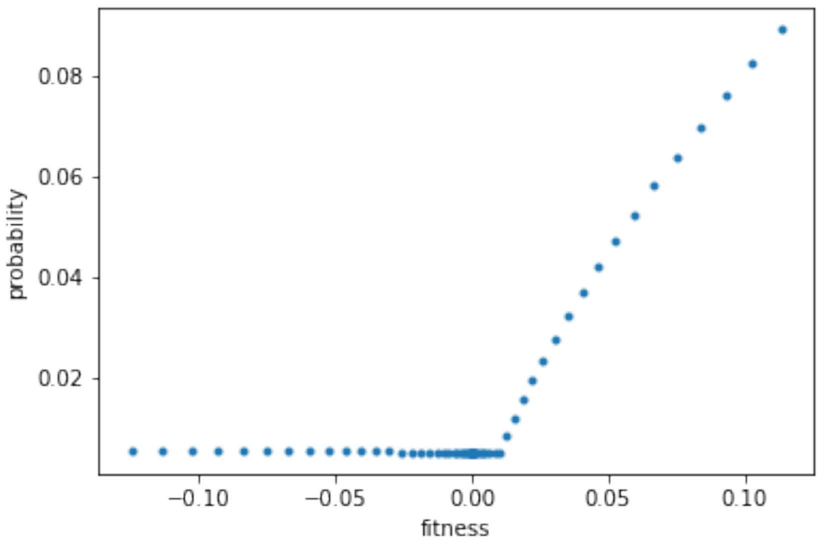


In [19]:
```
N = 64
aa = step(N, minval=0, maxval=N, step=1)
test_function(N, aa,'(x/N-1/2)**3', lambda x: (x/N-1/2)**3, 45)
```

function	z & tz/N	bad (f<%20)	good (f>%80)	excel (f>%90)
(x/N-1/2)**3	45 & 0.2812	0.064875	0.677118	0.438981

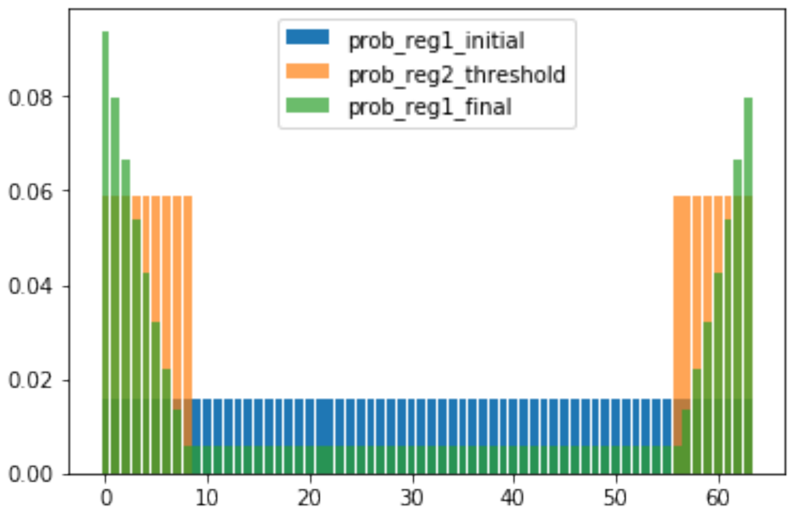


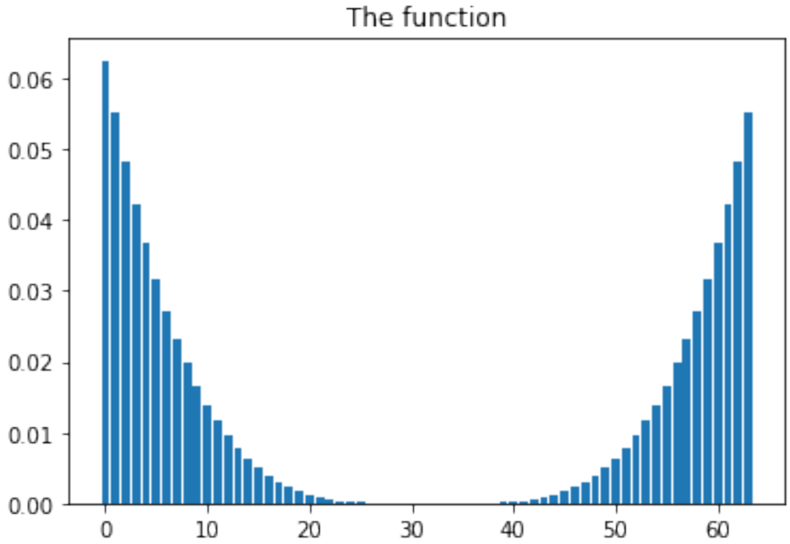


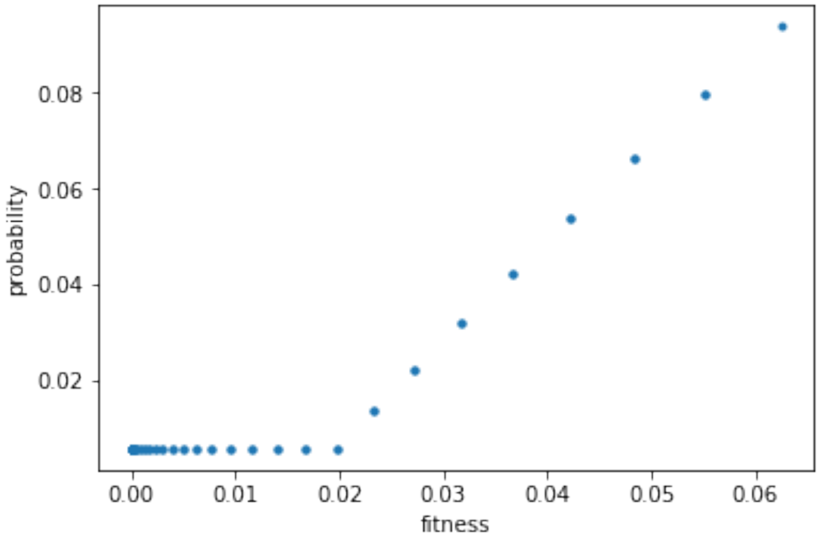


In [20]:
```
N = 64
aa = step(N, minval=0, maxval=N, step=1)
test_function(N, aa,'(x/N-1/2)**4', lambda x: (x/N-1/2)**4, 55)
```

```
| function      | z & tz/N      | bad (f<%20)  | good (f>%80) | exce
l (f>%90)|
| ------------ | ------------ | ------------ | ------------ | ----
-------- |
| (x/N-1/2)**4 |  55 & 0.2656 |    0.070574 |    0.664758 |
0.439607 |
```

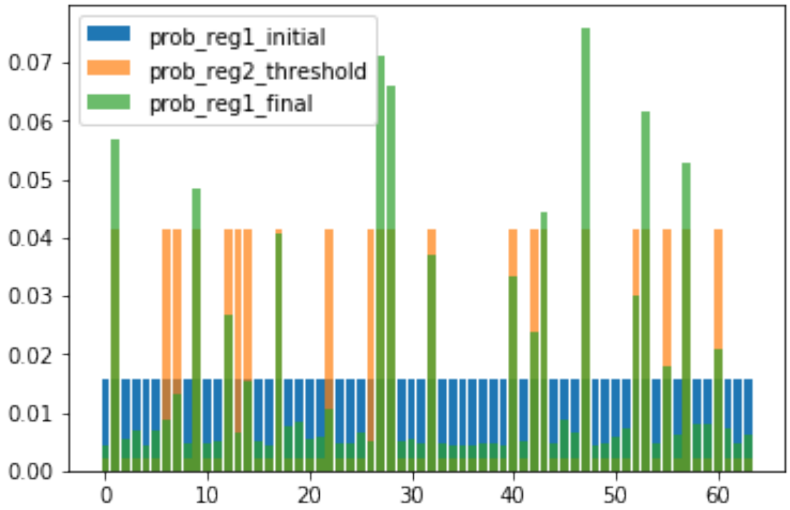


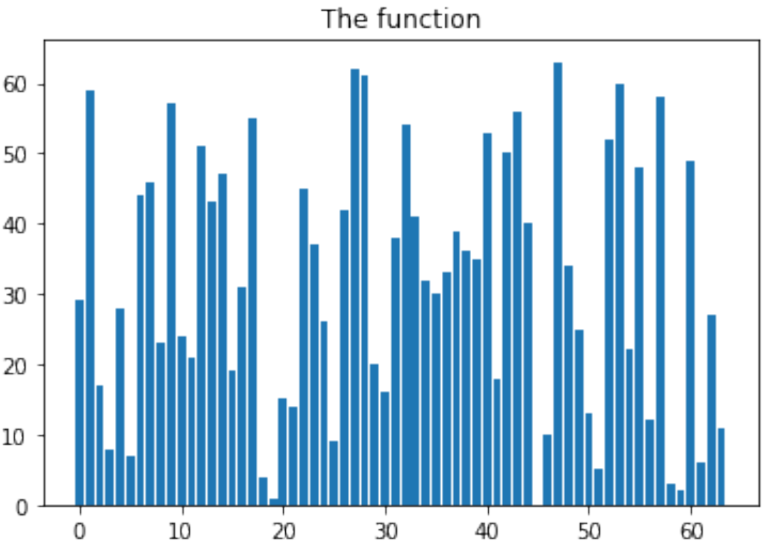


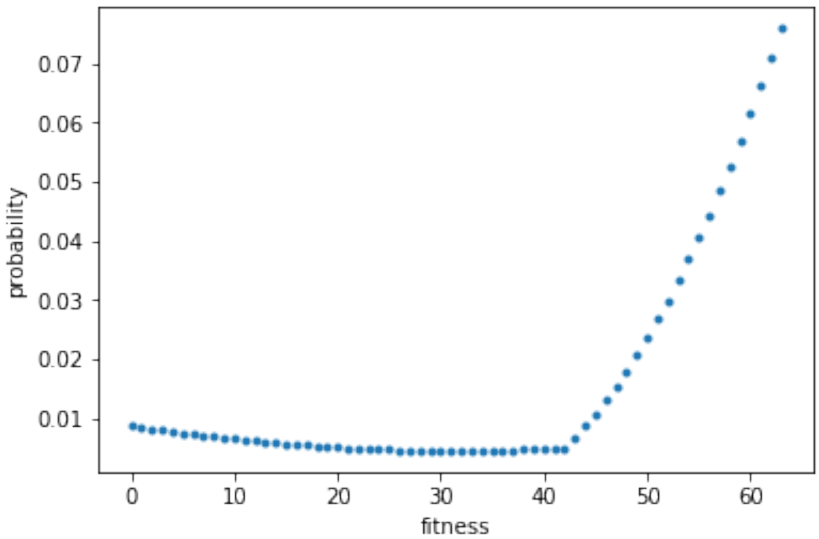


In [9]:
```
N = 64
aa = step(N, minval=0, maxval=
          N, step=1)
array = np.array(range(N))
np.random.seed(12)
np.random.shuffle(array)
test_function(N, aa,'unsorted', lambda x: array[x], 33)
```

```
| function     | z & tz/N     | bad (f<%20)  | good (f>%80) | exce
l (f>%90)|
| ------------ | ------------ | ------------ | ------------ | ----
-------- |
|     unsorted |  33 & 0.3438 |     0.088594 |     0.617464 |
0.384160 |
```


prob_reg1_initial
prob_reg2_threshold
prob_reg1_final
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
0
10
20
30
40
50
60


The function
60
50
40
30
20
10
0
0
10
20
30
40
50
60


probability
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
10
20
30
40
50
60
fitness

In [ ]: