In [2]:

```
import pandas as pd

def two_point_equal_interval_min():
    def f(x):
        return round((x**5 - 5*x**3 - 20*x + 5), 6)

    a = float(input("a = "))
    b = float(input("b = "))
    E = float(input("E = "))
    k = 0
    res = {"k":[], "a(k)":[], "b(k)":[],  "x1":[], "x2":[], "f(x1)":[], "f(x2)":[]}

    while abs(a - b) >= E:
        x1 = round(a + (b - a)/3, 7)
        x2 = round(b - (b - a)/3, 7)
        f1 = f(x1)
        f2 = f(x2)
        res["k"].append(k)
        res["a(k)"].append(a)
        res["b(k)"].append(b)
        res["x1"].append(x1)
        res["x2"].append(x2)
        res["f(x1)"].append(f1)
        res["f(x2)"].append(f2)
        if f1 >= f2:
            a = x1
        else:
            b = x2
        k += 1
    print(pd.DataFrame(res).to_string(index=False))

    if f(x1) <= f(x2):
        print(x1)
    else:
        print(x2)


two_point_equal_interval_min()   # Calling a function
```

```
a = 0
b = 3
E = 0.001
 k     a(k)     b(k)       x1       x2      f(x1)      f(x2)
 0 0.000000 3.000000 1.000000 2.000000 -19.000000 -43.000000
 1 1.000000 3.000000 1.666667 2.333333 -38.621400 -36.020578
 2 1.000000 2.333333 1.444444 1.888889 -32.669595 -42.429220
 3 1.444444 2.333333 1.740741 2.037037 -40.205140 -42.929616
 4 1.740741 2.333333 1.938272 2.135802 -42.817568 -41.986780
 5 1.740741 2.135802 1.872428 2.004115 -42.256320 -42.999151
 6 1.872428 2.135802 1.960219 2.048011 -42.923054 -42.880821
 7 1.872428 2.048011 1.930956 1.989483 -42.772938 -42.994510
 8 1.930956 2.048011 1.969974 2.008993 -42.955861 -42.995931
 9 1.969974 2.048011 1.995986 2.021999 -42.999197 -42.975428
10 1.969974 2.021999 1.987316 2.004657 -42.992026 -42.998912
11 1.987316 2.021999 1.998876 2.010438 -42.999937 -42.994513
12 1.987316 2.010438 1.995023 2.002730 -42.998766 -42.999627
13 1.995023 2.010438 2.000161 2.005299 -42.999999 -42.998591
14 1.995023 2.005299 1.998448 2.001874 -42.999880 -42.999824
15 1.995023 2.001874 1.997307 1.999590 -42.999638 -42.999992
16 1.997307 2.001874 1.998829 2.000351 -42.999931 -42.999994
17 1.998829 2.001874 1.999844 2.000859 -42.999999 -42.999963
18 1.998829 2.000859 1.999506 2.000182 -42.999988 -42.999998
19 1.999506 2.000859 1.999957 2.000408 -43.000000 -42.999992
1.9999567
```

In [3]:
```python
import pandas as pd

def two_point_equal_interval_max():
    def f(x):
        return round((x**5 - 5*x**3 - 20*x + 5), 6)

    a = float(input("a = "))
    b = float(input("b = "))
    E = float(input("E = "))
    k = 0
    res = {"k":[], "a(k)":[], "b(k)":[],  "x1":[], "x2":[], "f(x1)":[], "f(x2)":[]}

    while abs(a - b) >= E:
        x1 = round(a + (b - a)/3, 5)
        x2 = round(b - (b - a)/3, 5)
        f1 = f(x1)
        f2 = f(x2)
        res["k"].append(k)
        res["a(k)"].append(a)
        res["b(k)"].append(b)
        res["x1"].append(x1)
        res["x2"].append(x2)
        res["f(x1)"].append(f1)
        res["f(x2)"].append(f2)
        if f1 >= f2:
            b = x2
        else:
            a = x1
        k += 1
```

```
    print(pd.DataFrame(res).to_string(index=False))

    if f(x1) <= f(x2):
        print(x2)
    else:
        print(x1)
two_point_equal_interval_max()
```

```
a = -3
b = 0
E = 0.001
 k     a(k)     b(k)       x1       x2     f(x1)     f(x2)
 0 -3.00000  0.00000 -2.00000 -1.00000 53.000000 29.000000
 1 -3.00000 -1.00000 -2.33333 -1.66667 46.020731 48.621476
 2 -2.33333 -1.00000 -1.88889 -1.44444 52.429231 42.669465
 3 -2.33333 -1.44444 -2.03703 -1.74074 52.929643 50.205127
 4 -2.33333 -1.74074 -2.13580 -1.93827 51.986818 52.817559
 5 -2.13580 -1.74074 -2.00411 -1.87243 52.999153 52.256343
 6 -2.13580 -1.87243 -2.04801 -1.96022 52.880825 52.923056
 7 -2.04801 -1.87243 -1.98948 -1.93096 52.994507 52.772966
 8 -2.04801 -1.93096 -2.00899 -1.96998 52.995933 52.955879
 9 -2.04801 -1.96998 -2.02200 -1.99599 52.975425 52.999198
10 -2.02200 -1.96998 -2.00466 -1.98732 52.998911 52.992032
11 -2.02200 -1.98732 -2.01044 -1.99888 52.994510 52.999937
12 -2.01044 -1.98732 -2.00273 -1.99503 52.999627 52.998769
13 -2.01044 -1.99503 -2.00530 -2.00017 52.998590 52.999999
14 -2.00530 -1.99503 -2.00188 -1.99845 52.999823 52.999880
15 -2.00188 -1.99503 -1.99960 -1.99731 52.999992 52.999639
16 -2.00188 -1.99731 -2.00036 -1.99883 52.999994 52.999932
17 -2.00188 -1.99883 -2.00086 -1.99985 52.999963 52.999999
18 -2.00086 -1.99883 -2.00018 -1.99951 52.999998 52.999988
19 -2.00086 -1.99951 -2.00041 -1.99996 52.999992 53.000000
-1.99996
```

In [4]:

```
import pandas as pd

def bisection_method_min():
    def f(x):
        return round((x**5 - 5*x**3 - 20*x + 5), 6)
    a = float(input("a = "))
    b = float(input("b = "))
    while True:
        E = float(input("E = "))
        d = float(input("d = "))
        if E > d:
            break
        elif E <= 0:
            print("E must be greater than 0: ")
        else:
            print("E must be greater than d: ")
    k = 0
    res = {"k":[], "a(k)":[], "b(k)":[],  "x1":[], "x2":[], "f(x1)":[], "f(x2)":[]}

    while abs(a - b) >= E:
        x1 = round((a + b)/2 - d/2, 4)
        x2 = round((a + b)/2 + d/2, 4)
```

```
        f1 = f(x1)
        f2 = f(x2)
        res["k"].append(k)
        res["a(k)"].append(a)
        res["b(k)"].append(b)
        res["x1"].append(x1)
        res["x2"].append(x2)
        res["f(x1)"].append(f1)
        res["f(x2)"].append(f2)

        if f1 >= f2:
            a = x1
        else:
            b = x2
        k += 1
    print(pd.DataFrame(res).to_string(index=False))
    if f(x1) <= f(x2):
        print(x1)
    else:
        print(x2)

bisection_method_min()
```

```
a = 0
b = 3
E = 0.001
d = 0.00001
 k   a(k)   b(k)     x1     x2      f(x1)      f(x2)
 0 0.0000 3.0000 1.5000 1.5000 -34.281250 -34.281250
 1 1.5000 3.0000 2.2500 2.2500 -39.288086 -39.288086
 2 2.2500 3.0000 2.6250 2.6250 -13.302582 -13.302582
 3 2.6250 3.0000 2.8125 2.8125  13.493161  13.493161
 4 2.8125 3.0000 2.9062 2.9063  31.460653  31.481653
 5 2.8125 2.9063 2.8594 2.8594  22.067436  22.067436
 6 2.8594 2.9063 2.8828 2.8829  26.656536  26.676604
 7 2.8594 2.8829 2.8711 2.8712  24.335483  24.355096
 8 2.8594 2.8712 2.8653 2.8653  23.204573  23.204573
 9 2.8653 2.8712 2.8682 2.8683  23.768406  23.787906
10 2.8653 2.8683 2.8668 2.8668  23.495806  23.495806
11 2.8668 2.8683 2.8675 2.8676  23.632011  23.651485
2.8675
```

In [5]:
```
import pandas as pd

def bisection_method_max():
    def f(x):
        return round((x**5 - 5*x**3 - 20*x + 5), 6)
    a = float(input("a = "))
    b = float(input("b = "))
    while True:
        E = float(input("E = "))
        d = float(input("d = "))
        if E > d:
            break
        elif E <= 0:
            print("E must be greater than 0: ")
```

```
        else:
            print("E must be greater than d: ")
    k = 0
    res = {"k":[], "a(k)":[], "b(k)":[],  "x1":[], "x2":[], "f(x1)":[], "f(x2)":[]}

    while abs(a - b) >= E:
        x1 = round((a + b)/2 - d/2, 5)
        x2 = round((a + b)/2 + d/2, 5)
        f1 = f(x1)
        f2 = f(x2)
        res["k"].append(k)
        res["a(k)"].append(a)
        res["b(k)"].append(b)
        res["x1"].append(x1)
        res["x2"].append(x2)
        res["f(x1)"].append(f1)
        res["f(x2)"].append(f2)
        if f1 >= f2:
            b = x2
        else:
            a = x1
        k += 1
    print(pd.DataFrame(res).to_string(index=False))
    if f(x1) <= f(x2):
        print(x2)
    else:
        print(x1)
bisection_method_max()
```

```
a = -3
b = 0
E = 0.001
d = 0.0001
 k     a(k)     b(k)       x1       x2     f(x1)     f(x2)
 0 -3.00000  0.00000 -1.50005 -1.49995 44.282672 44.279828
 1 -3.00000 -1.49995 -2.25002 -2.24993 49.287442 49.290340
 2 -2.25002 -1.49995 -1.87504 -1.87494 52.285136 52.284042
 3 -2.25002 -1.87494 -2.06253 -2.06243 52.795789 52.796456
 4 -2.06253 -1.87494 -1.96879 -1.96869 52.952351 52.952049
 5 -2.06253 -1.96869 -2.01566 -2.01556 52.987603 52.987762
 6 -2.01566 -1.96869 -1.99223 -1.99212 52.996998 52.996912
 7 -2.01566 -1.99212 -2.00394 -2.00384 52.999222 52.999261
 8 -2.00394 -1.99212 -1.99808 -1.99798 52.999816 52.999796
 9 -2.00394 -1.99798 -2.00101 -2.00091 52.999949 52.999959
10 -2.00101 -1.99798 -1.99955 -1.99944 52.999990 52.999984
11 -2.00101 -1.99944 -2.00027 -2.00018 52.999996 52.999998
-2.00018
```

In [6]:

```
import pandas as pd

def F(n):
    if n < 0:
        print("wrong input ")
    elif n == 0:
        return 1
    elif n == 1:
```

```
        return 1
    else:
        return F(n - 1) + F(n - 2)


def fibonacci_method_min():
    def f(x):
        return round((x**5 - 5*x**3 - 20*x + 5), 6)

    a = float(input("a = "))
    b = float(input("b = "))
#     N = int(input("N = "))    # Here some code is missing for choosing N
#                               # N = 16 for E = 0.001
    E = float(input("E = "))
    i = 0
    while F(i) < 1/E:
        i += 1
    N = i

    L1 = b - a
    res = {"a(k)":[], "b(k)":[],  "x1":[], "x2":[], "f(x1)":[], "f(x2)":[], "L(k)":
    x1 = round(a + (F(N - 2)/F(N))*L1, 5)
    x2 = round(b - (F(N - 2)/F(N))*L1, 5)
    f1 = f(x1)
    f2 = f(x2)
    res["a(k)"].append(a)
    res["b(k)"].append(b)
    res["x1"].append(x1)
    res["x2"].append(x2)
    res["f(x1)"].append(f1)
    res["f(x2)"].append(f2)
    res["L(k)"].append(L1)
    for i in range(2, N):
        L = round((F(N - (i - 1)) / F(N)) * L1, 5)
        res["L(k)"].append(L)
        if f1 >= f2:
            a = x1
            x1 = x2
            f1 = f2
            x2 = round(b - (F(N - (i + 1))/F(N - (i - 1)))*L, 5)
            f2 = f(x2)
            res["a(k)"].append(a)
            res["b(k)"].append(b)
            res["x1"].append(x1)
            res["x2"].append(x2)
            res["f(x1)"].append(f1)
            res["f(x2)"].append(f2)
        else:
            b = x2
            x2 = x1
            f2 = f1
            x1 = round(a + (F(N - (i + 1))/F(N - (i - 1)))*L, 5)
            f1 = f(x1)
            res["a(k)"].append(a)
            res["b(k)"].append(b)
            res["x1"].append(x1)
```

```
            res["x2"].append(x2)
            res["f(x1)"].append(f1)
            res["f(x2)"].append(f2)

    print(pd.DataFrame(res))

    if f(x1) <= f(x2):
        print(x1)
    else:
        print(x2)

fibonacci_method_min()
```

```
a = 0
b = 3
E = 0.001
       a(k)     b(k)       x1       x2      f(x1)      f(x2)     L(k)
0   0.00000  3.00000  1.14590  1.85410 -23.465574 -42.039895  3.00000
1   1.14590  3.00000  1.85410  2.29180 -42.039895 -37.798413  1.85410
2   1.14590  2.29180  1.58360  1.85410 -36.569404 -42.039895  1.14590
3   1.58360  2.29180  1.85410  2.02129 -42.039895 -42.976997  0.70820
4   1.85410  2.29180  2.02129  2.12461 -42.976997 -42.153455  0.43770
5   1.85410  2.12461  1.95742  2.02129 -42.912016 -42.976997  0.27051
6   1.95742  2.12461  2.02129  2.06074 -42.976997 -42.807552  0.16719
7   1.95742  2.06074  1.99687  2.02129 -42.999511 -42.976997  0.10332
8   1.95742  2.02129  1.98184  1.99687 -42.983719 -42.999511  0.06387
9   1.98184  2.02129  1.99687  2.00626 -42.999511 -42.998032  0.03945
10  1.98184  2.00626  1.99123  1.99687 -42.996178 -42.999511  0.02442
11  1.99123  2.00626  1.99687  2.00062 -42.999511 -42.999981  0.01503
12  1.99687  2.00626  2.00062  2.00250 -42.999981 -42.999687  0.00939
13  1.99687  2.00250  1.99875  2.00062 -42.999922 -42.999981  0.00564
14  1.99875  2.00250  2.00062  2.00062 -42.999981 -42.999981  0.00376
2.00062
```

In [ ]:

```
def F(n):
    if n < 0:
        print("wrong input ")
    elif n == 0:
        return 1
    elif n == 1:
        return 1
    else:
        return F(n - 1) + F(n - 2)

E = 0.001
i = 0
while F(i) < 1/E:
    i += 1
N = i

print(N)
```

In [7]:

```
import pandas as pd

def F(n):
    if n < 0:
```

```
        print("wrong input ")
    elif n == 0:
        return 1
    elif n == 1:
        return 1
    else:
        return F(n - 1) + F(n - 2)

def fibonacci_method_max():
    def f(x):
        return round((x**5 - 5*x**3 - 20*x + 5), 6)

    a = float(input("a = "))
    b = float(input("b = "))
#     N = int(input("N = "))          # N = 16 for E = 0.001
    E = float(input("E = "))
    i = 0
    while F(i) < 1/E:
        i += 1
    N = i
    L1 = b - a
    res = {"a(k)":[], "b(k)":[],  "x1":[], "x2":[], "f(x1)":[], "f(x2)":[], "L(k)":
    x1 = round(a + (F(N - 2)/F(N))*L1, 5)
    x2 = round(b - (F(N - 2)/F(N))*L1, 5)
    f1 = f(x1)
    f2 = f(x2)
    res["a(k)"].append(a)
    res["b(k)"].append(b)
    res["x1"].append(x1)
    res["x2"].append(x2)
    res["f(x1)"].append(f1)
    res["f(x2)"].append(f2)
    res["L(k)"].append(L1)

    for i in range(2, N):
        L = round((F(N - (i - 1)) / F(N)) * L1, 5)
        res["L(k)"].append(L)
        if f1 >= f2:
            b = x2
            x2 = x1
            f2 = f1
            x1 = round(a + (F(N - (i + 1))/F(N - (i - 1)))*L, 5)
            f1 = f(x1)
            res["a(k)"].append(a)
            res["b(k)"].append(b)
            res["x1"].append(x1)
            res["x2"].append(x2)
            res["f(x1)"].append(f1)
            res["f(x2)"].append(f2)
        else:
            a = x1
            x1 = x2
            f1 = f2
            x2 = round(b - (F(N - (i + 1))/F(N - (i - 1)))*L, 5)
            f2 = f(x2)
            res["a(k)"].append(a)
```

```
            res["b(k)"].append(b)
            res["x1"].append(x1)
            res["x2"].append(x2)
            res["f(x1)"].append(f1)
            res["f(x2)"].append(f2)

    print(pd.DataFrame(res))

    if f(x1) <= f(x2):
        print(x2)
    else:
        print(x1)

fibonacci_method_max()
```

```
a = -3
b = 0
E = 0.001
       a(k)     b(k)       x1       x2      f(x1)      f(x2)     L(k)
0  -3.00000  0.00000 -1.85410 -1.14590  52.039895  33.465574  3.00000
1  -3.00000 -1.14590 -2.29180 -1.85410  47.798413  52.039895  1.85410
2  -2.29180 -1.14590 -1.85410 -1.58360  52.039895  46.569404  1.14590
3  -2.29180 -1.58360 -2.02129 -1.85410  52.976997  52.039895  0.70820
4  -2.29180 -1.85410 -2.12461 -2.02129  52.153455  52.976997  0.43770
5  -2.12461 -1.85410 -2.02129 -1.95742  52.976997  52.912016  0.27051
6  -2.12461 -1.95742 -2.06074 -2.02129  52.807552  52.976997  0.16719
7  -2.06074 -1.95742 -2.02129 -1.99687  52.976997  52.999511  0.10332
8  -2.02129 -1.95742 -1.99687 -1.98184  52.999511  52.983719  0.06387
9  -2.02129 -1.98184 -2.00626 -1.99687  52.998032  52.999511  0.03945
10 -2.00626 -1.98184 -1.99687 -1.99123  52.999511  52.996178  0.02442
11 -2.00626 -1.99123 -2.00062 -1.99687  52.999981  52.999511  0.01503
12 -2.00626 -1.99687 -2.00250 -2.00062  52.999687  52.999981  0.00939
13 -2.00250 -1.99687 -2.00062 -1.99875  52.999981  52.999922  0.00564
14 -2.00250 -1.99875 -2.00062 -2.00062  52.999981  52.999981  0.00376
-2.00062
```

In [8]:
```
import pandas as pd

def golden_ratio_method_min():
    def f(x):
        return round((x**5 - 5*x**3 - 20*x + 5), 6)

    a = float(input("a = "))
    b = float(input("b = "))
    while True:
        E = float(input("E = "))
        if E > 0:
            break
        else:
            print("E must be greater than 0 ")
    res = {"a(k)":[], "b(k)":[],  "x1":[], "x2":[], "f(x1)":[], "f(x2)":[]}
    x1 = round(a + (1/2.618)*(b - a), 5)
    x2 = round(b - (1/2.618)*(b - a), 5)
    f1 = f(x1)
    f2 = f(x2)
    res["a(k)"].append(a)
```

```
    res["b(k)"].append(b)
    res["x1"].append(x1)
    res["x2"].append(x2)
    res["f(x1)"].append(f1)
    res["f(x2)"].append(f2)

    while abs(a - b) > E:
        if f1 >= f2:
            a = x1
            x1 = x2
            f1 = f2
            x2 = round(b - (1/2.618)*(b - a), 5)
            f2 = f(x2)
            res["a(k)"].append(a)
            res["b(k)"].append(b)
            res["x1"].append(x1)
            res["x2"].append(x2)
            res["f(x1)"].append(f1)
            res["f(x2)"].append(f2)
        else:
            b = x2
            x2 = x1
            f2 = f1
            x1 = round(a + (1/2.618)*(b - a), 5)
            f1 = f(x1)
            res["a(k)"].append(a)
            res["b(k)"].append(b)
            res["x1"].append(x1)
            res["x2"].append(x2)
            res["f(x1)"].append(f1)
            res["f(x2)"].append(f2)

    print(pd.DataFrame(res))

    if f(x1) <= f(x2):
        print(x1)
    else:
        print(x2)
golden_ratio_method_min()
```

```
a = 0
b = 3
E = 0.001
       a(k)     b(k)       x1       x2      f(x1)      f(x2)
0   0.00000  3.00000  1.14591  1.85409 -23.465884 -42.039771
1   1.14591  3.00000  1.85409  2.29179 -42.039771 -37.798805
2   1.14591  2.29179  1.58360  1.85409 -36.569404 -42.039771
3   1.58360  2.29179  1.85409  2.02128 -42.039771 -42.977019
4   1.85409  2.29179  2.02128  2.12460 -42.977019 -42.153596
5   1.85409  2.12460  1.95742  2.02128 -42.912016 -42.977019
6   1.95742  2.12460  2.02128  2.06074 -42.977019 -42.807552
7   1.95742  2.06074  1.99689  2.02128 -42.999517 -42.977019
8   1.95742  2.02128  1.98181  1.99689 -42.983666 -42.999517
9   1.98181  2.02128  1.99689  2.00620 -42.999517 -42.998070
10  1.98181  2.00620  1.99113  1.99689 -42.996091 -42.999517
11  1.99113  2.00620  1.99689  2.00044 -42.999517 -42.999990
12  1.99689  2.00620  2.00044  2.00264 -42.999990 -42.999651
13  1.99689  2.00264  1.99909  2.00044 -42.999959 -42.999990
14  1.99909  2.00264  2.00044  2.00128 -42.999990 -42.999918
15  1.99909  2.00128  1.99993  2.00044 -43.000000 -42.999990
16  1.99909  2.00044  1.99961  1.99993 -42.999992 -43.000000
17  1.99961  2.00044  1.99993  2.00012 -43.000000 -42.999999
1.99993
```

In [9]:

```
import pandas as pd

def golden_ratio_method_max():
    def f(x):
        return round((x**5 - 5*x**3 - 20*x + 5), 6)

    a = float(input("a = "))
    b = float(input("b = "))
    while True:
        E = float(input("E = "))
        if E > 0:
            break
        else:
            print("E must be greater than 0 ")
    res = {"a(k)":[], "b(k)":[],  "x1":[], "x2":[], "f(x1)":[], "f(x2)":[]}
    x1 = round(a + (1/2.618)*(b - a), 5)
    x2 = round(b - (1/2.618)*(b - a), 5)
    f1 = f(x1)
    f2 = f(x2)
    res["a(k)"].append(a)
    res["b(k)"].append(b)
    res["x1"].append(x1)
    res["x2"].append(x2)
    res["f(x1)"].append(f1)
    res["f(x2)"].append(f2)

    while abs(a - b) > E:
        if f1 >= f2:
            b = x2
            x2 = x1
            f2 = f1
            x1 = round(a + (1/2.618)*(b - a), 5)
```

```
            f1 = f(x1)
            res["a(k)"].append(a)
            res["b(k)"].append(b)
            res["x1"].append(x1)
            res["x2"].append(x2)
            res["f(x1)"].append(f1)
            res["f(x2)"].append(f2)
        else:
            a = x1
            x1 = x2
            f1 = f2
            x2 = round(b - (1/2.618)*(b - a), 5)
            f2 = f(x2)
            res["a(k)"].append(a)
            res["b(k)"].append(b)
            res["x1"].append(x1)
            res["x2"].append(x2)
            res["f(x1)"].append(f1)
            res["f(x2)"].append(f2)

    print(pd.DataFrame(res))

    if f(x1) <= f(x2):
        print(x1)
    else:
        print(x2)

golden_ratio_method_max()
```

```
a = -3
b = 0
E = 0.001
       a(k)     b(k)       x1       x2      f(x1)      f(x2)
0  -3.00000  0.00000 -1.85409 -1.14591  52.039771  33.465884
1  -3.00000 -1.14591 -2.29179 -1.85409  47.798805  52.039771
2  -2.29179 -1.14591 -1.85409 -1.58360  52.039771  46.569404
3  -2.29179 -1.58360 -2.02128 -1.85409  52.977019  52.039771
4  -2.29179 -1.85409 -2.12460 -2.02128  52.153596  52.977019
5  -2.12460 -1.85409 -2.02128 -1.95742  52.977019  52.912016
6  -2.12460 -1.95742 -2.06074 -2.02128  52.807552  52.977019
7  -2.06074 -1.95742 -2.02128 -1.99689  52.977019  52.999517
8  -2.02128 -1.95742 -1.99689 -1.98181  52.999517  52.983666
9  -2.02128 -1.98181 -2.00620 -1.99689  52.998070  52.999517
10 -2.00620 -1.98181 -1.99689 -1.99113  52.999517  52.996091
11 -2.00620 -1.99113 -2.00044 -1.99689  52.999990  52.999517
12 -2.00620 -1.99689 -2.00264 -2.00044  52.999651  52.999990
13 -2.00264 -1.99689 -2.00044 -1.99909  52.999990  52.999959
14 -2.00264 -1.99909 -2.00128 -2.00044  52.999918  52.999990
15 -2.00128 -1.99909 -2.00044 -1.99993  52.999990  53.000000
16 -2.00044 -1.99909 -1.99993 -1.99961  53.000000  52.999992
17 -2.00044 -1.99961 -2.00012 -1.99993  52.999999  53.000000
-2.00012
```

In [12]:

```
import pandas as pd

def quadratic_interpolation_min():
```

```python
def f(x):
    return round((x**5 - 5*x**3 - 20*x + 5), 6)
a = float(input("a = "))
b = float(input("b = "))
while True:
    E = float(input("E = "))
    if E > 0:
        break
    else:
        print("E must be greater than 0 ")
c = a + (b - a)/2 + 0.1
fa = f(a)
fb = f(b)
fc = f(c)
res = {"a(k)":[], "c(k)":[], "b(k)":[], "xstar":[],  "fa":[], "fc":[], "fb":[],
while True:
    # fold = fc
    xstar = round((1/2)*(((b**2 - c**2)*fa + (c**2 - a**2)*fb + (a**2 - b**2)*f
    fx = f(xstar)
    res["a(k)"].append(a)
    res["b(k)"].append(b)
    res["c(k)"].append(c)
    res["xstar"].append(xstar)
    res["fa"].append(fa)
    res["fb"].append(fb)
    res["fc"].append(fc)
    res["fx"].append(fx)

    # print(pd.DataFrame(res))
    if xstar < c and fx <= fc:
        b = c
        c = xstar
        fb = fc
        fc = fx
    elif xstar < c and fx >= fc:
        a = xstar
        fa = fx
    elif xstar > c and fx <= fc:
        a = c
        c = xstar
        fa = fc
        fc = fx
    elif xstar > c and fx >= fc:
        b = xstar
        fb = fx

    # if abs(b - a) < E or abs((fold - fc)/fold < E):
    if c == xstar:
        c += 0.00001
    if abs(a - b) < E:
        res["a(k)"].append(a)
        res["b(k)"].append(b)
        res["c(k)"].append(c)
        res["xstar"].append(xstar)
        res["fa"].append(fa)
        res["fb"].append(fb)
```

```
            res["fc"].append(fc)
            res["fx"].append(fx)
            break
    return (pd.DataFrame(res))
quadratic_interpolation_min()
```

```
a = 0
b = 3
E = 0.001
```

Out[12]:

	a(k)	c(k)	b(k)	xstar	fa	fc	fb	fx
0	0.000000	1.600000	3.00000	1.234889	5.000000	-36.994240	53.0	-26.241802
1	1.234889	1.600000	3.00000	1.694738	-26.241802	-36.994240	53.0	-39.252175
2	1.600000	1.694748	3.00000	1.823883	-36.994240	-39.252175	53.0	-41.630882
3	1.694748	1.823893	3.00000	1.880888	-39.252175	-41.630882	53.0	-42.347775
4	1.823893	1.880898	3.00000	1.928031	-41.630882	-42.347775	53.0	-42.753804
5	1.880898	1.928041	3.00000	1.953676	-42.347775	-42.753804	53.0	-42.896138
6	1.928041	1.953686	3.00000	1.971468	-42.753804	-42.896138	53.0	-42.960103
7	1.953686	1.971478	3.00000	1.981993	-42.896138	-42.960103	53.0	-42.983991
8	1.971478	1.982003	3.00000	1.988829	-42.960103	-42.983991	53.0	-42.993809
9	1.982003	1.988839	3.00000	1.993007	-42.983991	-42.993809	53.0	-42.997567
10	1.988839	1.993017	3.00000	1.995654	-42.993809	-42.997567	53.0	-42.999058
11	1.993017	1.995664	3.00000	1.997290	-42.997567	-42.999058	53.0	-42.999633
12	1.995664	1.997300	3.00000	1.998319	-42.999058	-42.999633	53.0	-42.999859
13	1.997300	1.998329	3.00000	1.998961	-42.999633	-42.999859	53.0	-42.999946
14	1.998329	1.998971	3.00000	1.999357	-42.999859	-42.999946	53.0	-42.999979
15	1.998971	1.999367	3.00000	1.999603	-42.999946	-42.999979	53.0	-42.999992
16	1.999367	1.999613	3.00000	1.999765	-42.999979	-42.999992	53.0	-42.999997
17	1.999613	1.999775	3.00000	1.999855	-42.999992	-42.999997	53.0	-42.999999
18	1.999775	1.999865	3.00000	1.999936	-42.999997	-42.999999	53.0	-43.000000
19	1.999865	1.999946	3.00000	1.999970	-42.999999	-43.000000	53.0	-43.000000
20	1.999946	1.999980	3.00000	1.999963	-43.000000	-43.000000	53.0	-43.000000
21	1.999946	1.999973	1.99998	1.999963	-43.000000	-43.000000	-43.0	-43.000000

In [15]:

```
import pandas as pd

def quadratic_interpolation_max():
    def f(x):
        return round((x**5 - 5*x**3 - 20*x + 5), 6)

    a = float(input("a = "))
    b = float(input("b = "))
```

```
    while True:
        E = float(input("E = "))
        if E > 0:
            break
        else:
            print("E must be greater than 0 ")
    c = a + (b - a)/2 + 0.1
    fa = f(a)
    fb = f(b)
    fc = f(c)
    res = {"a(k)":[], "c(k)":[], "b(k)":[], "xstar":[],  "fa":[], "fc":[], "fb":[],
    while True:
        # fold = fc
        xstar = round((1/2)*(((b**2 - c**2)*fa + (c**2 - a**2)*fb + (a**2 - b**2)*f
        fx = f(xstar)
        res["a(k)"].append(a)
        res["b(k)"].append(b)
        res["c(k)"].append(c)
        res["xstar"].append(xstar)
        res["fa"].append(fa)
        res["fb"].append(fb)
        res["fc"].append(fc)
        res["fx"].append(fx)
        # print(pd.DataFrame(res))
        # print(pd.DataFrame(res))
        if xstar < c and fx <= fc:
            a = xstar
            fa = fx
        elif xstar < c and fx >= fc:
            b = c
            c = xstar
            fb = fc
            fc = fx
        elif xstar > c and fx <= fc:
            b = xstar
            fb = fx
        elif xstar > c and fx >= fc:
            a = c
            c = xstar
            fa = fc
            fc = fx

        # if abs(b - a) < E or abs((fold - fc)/fold < E):
        if c == xstar:
            c += 0.00001
        if abs(a - b) < E:
            res["a(k)"].append(a)
            res["b(k)"].append(b)
            res["c(k)"].append(c)
            res["xstar"].append(xstar)
            res["fa"].append(fa)
            res["fb"].append(fb)
            res["fc"].append(fc)
            res["fx"].append(fx)
            break
    return (pd.DataFrame(res))
```

```
quadratic_interpolation_max()
```

```
a = -3
b = 0
E = 0.001
```

Out[15]:

	a(k)	c(k)	b(k)	xstar	fa	fc	fb	fx
0	-3.000000	-1.400000	0.000000	-1.194936	-43.0	41.341760	5.000000	34.993541
1	-3.000000	-1.400000	-1.194936	-1.631394	-43.0	41.341760	34.993541	47.781570
2	-3.000000	-1.631384	-1.400000	-1.752148	-43.0	47.781570	41.341760	50.424567
3	-3.000000	-1.752138	-1.631384	-1.846562	-43.0	50.424567	47.781570	51.943816
4	-3.000000	-1.846552	-1.752138	-1.901372	-43.0	51.943816	50.424567	52.546268
5	-3.000000	-1.901362	-1.846552	-1.938669	-43.0	52.546268	51.943816	52.819859
6	-3.000000	-1.938659	-1.901362	-1.961289	-43.0	52.819859	52.546268	52.927081
7	-3.000000	-1.961279	-1.938659	-1.975877	-43.0	52.927081	52.819859	52.971392
8	-3.000000	-1.975867	-1.961279	-1.984879	-43.0	52.971392	52.927081	52.988688
9	-3.000000	-1.984869	-1.975867	-1.990566	-43.0	52.988688	52.971392	52.995579
10	-3.000000	-1.990556	-1.984869	-1.994098	-43.0	52.995579	52.988688	52.998266
11	-3.000000	-1.994088	-1.990556	-1.996314	-43.0	52.998266	52.995579	52.999322
12	-3.000000	-1.996304	-1.994088	-1.997689	-43.0	52.999322	52.998266	52.999733
13	-3.000000	-1.997679	-1.996304	-1.998553	-43.0	52.999733	52.999322	52.999895
14	-3.000000	-1.998543	-1.997679	-1.999089	-43.0	52.999895	52.999733	52.999959
15	-3.000000	-1.999079	-1.998543	-1.999434	-43.0	52.999959	52.999895	52.999984
16	-3.000000	-1.999424	-1.999079	-1.999629	-43.0	52.999984	52.999959	52.999993
17	-3.000000	-1.999619	-1.999424	-1.999762	-43.0	52.999993	52.999984	52.999997
18	-3.000000	-1.999752	-1.999619	-1.999842	-43.0	52.999997	52.999993	52.999999
19	-3.000000	-1.999832	-1.999752	-1.999922	-43.0	52.999999	52.999997	53.000000
20	-3.000000	-1.999912	-1.999832	-1.999937	-43.0	53.000000	52.999999	53.000000
21	-1.999937	-1.999912	-1.999832	-1.999937	53.0	53.000000	52.999999	53.000000

In [16]:

```
def newton_raphson():
    def fd(x):
        return round(3*x**2 - 12*x + 11, 6)
    def fdd(x):
        return round(6*x - 12, 6)
    a = 1
    b = 3
    E = 0.001
    k = 0
    x = 1
    res = {"k":[], "x_k":[], "x_k+1":[]}
```

```
    while True:
        x1 = round(x - fd(x)/fdd(x), 5)
        print(x1)
        k += 1
        if abs(x1 - x) >= E:
            x = x1
        else:
            break
newton_raphson()
```

```
1.33333
1.41667
1.42262
1.42265
```

In [17]:
```
def steepest_descentQ_min():
    import numpy as np
    import pandas as pd

    Q = np.array([[2, 1], [1, 4]])
    E = 0.001
    x_0 = np.array([[10, -10]]).T
    res = {"a0":[], "a1":[],  "g0":[], "g1":[]}

    def gradient(x):
        return np.dot(Q, x)

    while True:
        g = gradient(x_0)

        res["a0"].append(f'%.{6}f' % x_0[0][0])
        res["a1"].append(f'%.{6}f' % x_0[1][0])
        res["g0"].append(f'%.{6}f' % g[0][0])
        res["g1"].append(f'%.{6}f' % g[1][0])
        alpha = np.dot(g.T, g) / np.dot(g.T, np.dot(Q, g))
        x_1 = x_0 - alpha*g

        if abs(x_0[0] - x_1[0]) < E and abs(x_0[1] - x_1[1]) < E or (g[0][0] == 0 a
            break
        else:
            x_0 = x_1

    return pd.DataFrame(res)

steepest_descentQ_min()
```

Out[17]:

	a0	a1	g0	g1
0	10.000000	-10.000000	10.000000	-30.000000
1	6.875000	-0.625000	13.125000	4.375000
2	2.187500	-2.187500	2.187500	-6.562500
3	1.503906	-0.136719	2.871094	0.957031
4	0.478516	-0.478516	0.478516	-1.435547
5	0.328979	-0.029907	0.628052	0.209351
6	0.104675	-0.104675	0.104675	-0.314026
7	0.071964	-0.006542	0.137386	0.045795
8	0.022898	-0.022898	0.022898	-0.068693
9	0.015742	-0.001431	0.030053	0.010018
10	0.005009	-0.005009	0.005009	-0.015027
11	0.003444	-0.000313	0.006574	0.002191
12	0.001096	-0.001096	0.001096	-0.003287
13	0.000753	-0.000068	0.001438	0.000479

In [18]:

```
def steepest_descent_min():
    import sympy as sp
    import numpy as np
    import pandas as pd
    import math

    x, y = sp.symbols('x y')
    t = sp.symbols('t')
    e = math.e
    # f = e**(x-1) + e**(-y + 1) + (x - y)**2
    f = x**2 + x*y + 2*y**2

    print("f = ",f)

    def gradient(x0, x1):
        g = [[f.diff(x).evalf(subs = {x:x0, y:x1}), f.diff(y).evalf(subs = {x:x0, y
        return (np.array(g)).T

    def Hessian(x0, x1):
        g = [f.diff(x), f.diff(y)]

        H = [[g[0].diff(x), g[1].diff(x)],
             [g[0].diff(y), g[1].diff(y)]]

        Hv = [[H[0][0].evalf(subs = {x:x0, y:x1}), H[0][1].evalf(subs = {x:x0, y:x1
              [H[1][0].evalf(subs = {x:x0, y:x1}), H[1][1].evalf(subs = {x:x0, y:x1

        return np.array(Hv)

    x_0 = np.array([[10, -10]]).T
```

```
    Ea = 0.001
    res = {"a0":[], "a1":[],  "g0":[], "g1":[]}


    while True:
        g = gradient(x_0[0][0], x_0[1][0])
        H = Hessian(x_0[0][0], x_0[1][0])

        res["a0"].append(f'%.{6}f' % x_0[0][0])
        res["a1"].append(f'%.{6}f' % x_0[1][0])
        res["g0"].append(f'%.{6}f' % g[0][0])
        res["g1"].append(f'%.{6}f' % g[1][0])

        alpha = np.dot(g.T, g) / np.dot(g.T, np.dot(H, g))

        x_1 = x_0 - alpha*g

        if abs(x_0[0] - x_1[0]) < Ea and abs(x_0[1] - x_1[1]) < Ea or (g[0][0] == 0
            break
        else:
            x_0 = x_1

    print(pd.DataFrame(res))

steepest_descent_min()
```

```
f =  x**2 + x*y + 2*y**2
           a0          a1         g0          g1
0   10.000000  -10.000000  10.000000  -30.000000
1    6.875000   -0.625000  13.125000    4.375000
2    2.187500   -2.187500   2.187500   -6.562500
3    1.503906   -0.136719   2.871094    0.957031
4    0.478516   -0.478516   0.478516   -1.435547
5    0.328979   -0.029907   0.628052    0.209351
6    0.104675   -0.104675   0.104675   -0.314026
7    0.071964   -0.006542   0.137386    0.045795
8    0.022898   -0.022898   0.022898   -0.068693
9    0.015742   -0.001431   0.030053    0.010018
10   0.005009   -0.005009   0.005009   -0.015027
11   0.003444   -0.000313   0.006574    0.002191
12   0.001096   -0.001096   0.001096   -0.003287
13   0.000753   -0.000068   0.001438    0.000479
```

In [19]:

```
def cg_method_min():
    import sympy as sp
    import numpy as np
    import pandas as pd
    import math

    x, y = sp.symbols('x y')
    t = sp.symbols('t')
    e = math.e
    # f = e**(x-1) + e**(-y + 1) + (x - y)**2
    f = x**2 + x*y + 2*y**2
    print(f)
```

```
    def gradient(x0, x1):
        g = [[f.diff(x).evalf(subs = {x:x0, y:x1}), f.diff(y).evalf(subs = {x:x0, y
        return (np.array(g)).T

    def Hessian(x0, x1):
        g = [f.diff(x), f.diff(y)]

        H = [[g[0].diff(x), g[1].diff(x)],
             [g[0].diff(y), g[1].diff(y)]]

        Hv = [[H[0][0].evalf(subs = {x:x0, y:x1}), H[0][1].evalf(subs = {x:x0, y:x1
              [H[1][0].evalf(subs = {x:x0, y:x1}), H[1][1].evalf(subs = {x:x0, y:x1

        return np.array(Hv)

    x_0 = np.array([[10, -10]]).T
    E = 0.001
    n = 2
    res = {"a0":[], "a1":[],  "g0":[], "g1":[]}

#     H = Hessian(x_0[0][0], x_0[1][0])

    while True:
        g_0 = gradient(x_0[0][0], x_0[1][0])
        H = Hessian(x_0[0][0], x_0[1][0])
        # print(g_0)
        d = -g_0

        for _ in range(n):

            alpha = np.dot(g_0.T, g_0) / np.dot(d.T ,np.dot(H, d))

            x_1 = x_0 + alpha*d
            # print(x_1)
            g_1 = gradient(x_1[0][0], x_1[1][0])

            d = -g_1 + (np.dot(g_1.T, g_1) / np.dot(g_0.T, g_0))*d

            res["a0"].append(f'%.{6}f' % x_0[0][0])
            res["a1"].append(f'%.{6}f' % x_0[1][0])
            res["g0"].append(f'%.{6}f' % g_0[0][0])
            res["g1"].append(f'%.{6}f' % g_0[1][0])

            temp = x_0
            x_0 = x_1
            g_0 = g_1

        if abs(x_1[0][0] - temp[0][0]) < E and abs(x_1[1][0] - temp[1][0]) < E or (
#         if (((x_1 - temp)[0][0]**2 + (x_1 - temp)[1][0]**2)**0.5 < E) or (((g_0)[
            res["a0"].append(f'%.{6}f' % x_0[0][0])
            res["a1"].append(f'%.{6}f' % x_0[1][0])
            res["g0"].append(f'%.{6}f' % g_0[0][0])
            res["g1"].append(f'%.{6}f' % g_0[1][0])
            break
```

```
    print(pd.DataFrame(res))
    print(x_1, "is optimum\n")

cg_method_min()
```

```
x**2 + x*y + 2*y**2
          a0         a1         g0         g1
0  10.000000 -10.000000  10.000000 -30.000000
1   6.875000  -0.625000  13.125000   4.375000
2   0.000000   0.000000   0.000000   0.000000
[[0]
 [0]] is optimum
```

In [20]:

```
def newtons_min():
    import sympy as sp
    import numpy as np
    import pandas as pd
    import math

    x, y = sp.symbols('x y')
    t = sp.symbols('t')
    e = math.e
#     f = e**(x-1) + e**(-y + 1) + (x - y)**2
    f = x**2 + x*y + 2*y**2

    print(f)

    def gradient(x0, x1):
        g = [[f.diff(x).evalf(subs = {x:x0, y:x1}), f.diff(y).evalf(subs = {x:x0, y
        return (np.array(g)).T

    def Hessian(x0, x1):
        g = [f.diff(x), f.diff(y)]

        H = [[g[0].diff(x), g[1].diff(x)],
             [g[0].diff(y), g[1].diff(y)]]

        Hv = [[H[0][0].evalf(subs = {x:x0, y:x1}), H[0][1].evalf(subs = {x:x0, y:x1
              [H[1][0].evalf(subs = {x:x0, y:x1}), H[1][1].evalf(subs = {x:x0, y:x1

        return np.array(Hv)

    def fv(val):
        evalute = f.evalf(subs = {x:val[0][0], y:val[1][0]})
        return evalute

    def sol(H_0, g_0):
        h1, h2 = sp.symbols(('h1 h2'))
        h = np.array([[h1], [h2]])
        Hh = np.dot(H_0, h)
        eq1 = sp.Eq(Hh[0][0], g_0[0][0])
        eq2 = sp.Eq(Hh[1][0], g_0[1][0])
        R = sp.solve((eq1, eq2), (h1, h2))
        h = [[R[h1]], [R[h2]]]
```

```
        return np.array(h)


    x_0 = np.array([[10, -10]]).T
    E = 0.001
    res = {"a0":[], "a1":[], "g0":[], "g1":[]}


    while True:
        g_0 = gradient(x_0[0][0], x_0[1][0])

        res["a0"].append(f'%.{6}f' % x_0[0][0])
        res["a1"].append(f'%.{6}f' % x_0[1][0])
        res["g0"].append(f'%.{6}f' % g_0[0][0])
        res["g1"].append(f'%.{6}f' % g_0[1][0])

        H_0 = Hessian(x_0[0][0], x_0[1][0])

        h = sol(H_0, g_0)

        x_1 = x_0 - h

#         if (((x_1 - x_0)[0][0]**2 + (x_1 - x_0)[1][0]**2)**0.5 < E) or (((g_0)[0]
        if abs(x_1[0][0] - x_0[0][0]) < E and abs(x_1[1][0] - x_0[1][0]) < E or (((

            print(x_1, "is optimum\n")
            break
        else:
            x_0 = x_1

    print(pd.DataFrame(res))

newtons_min()
```

```
x**2 + x*y + 2*y**2
[[0]
 [0]] is optimum

          a0          a1         g0          g1
0  10.000000  -10.000000  10.000000  -30.000000
1   0.000000    0.000000   0.000000    0.000000
```

In [24]:

```
def marquadrt_min():
    import sympy as sp
    import numpy as np
    import pandas as pd
    import math

    x, y = sp.symbols('x y')
    t = sp.symbols('t')
    e = math.e
    # f = e**(x-1) + e**(-y + 1) + (x - y)**2
#     f = x - y + 2*x**2 + 2*x*y + y**2
    f = x**2 + x*y + 2*y**2

    print(f)
```

```python
def gradient(x0, x1):
    g = [[f.diff(x).evalf(subs = {x:x0, y:x1}), f.diff(y).evalf(subs = {x:x0, y
    return (np.array(g)).T

def Hessian(x0, x1):
    g = [f.diff(x), f.diff(y)]

    H = [[g[0].diff(x), g[1].diff(x)],
         [g[0].diff(y), g[1].diff(y)]]

    Hv = [[H[0][0].evalf(subs = {x:x0, y:x1}), H[0][1].evalf(subs = {x:x0, y:x1
          [H[1][0].evalf(subs = {x:x0, y:x1}), H[1][1].evalf(subs = {x:x0, y:x1

    return np.array(Hv)

def fv(val):
    evalute = f.evalf(subs = {x:val[0][0], y:val[1][0]})
    return evalute

def sol(H_0, g_0):
    h1, h2 = sp.symbols(('h1 h2'))
    h = np.array([[h1], [h2]])
    Hh = np.dot(H_0, h)
    eq1 = sp.Eq(Hh[0][0], g_0[0][0])
    eq2 = sp.Eq(Hh[1][0], g_0[1][0])
    R = sp.solve((eq1, eq2), (h1, h2))
    h = [[R[h1]], [R[h2]]]
    return np.array(h)


x_0 = np.array([[10, -10]]).T
E = 0.001
delta = 10000
c1 = 1/4
c2 = 2
res = {"a0":[], "a1":[], "delta":[], "g0":[], "g1":[], "f0":[], "f1":[]}

while True:
    g = gradient(x_0[0][0], x_0[1][0])
    if (g[0][0]**2 + g[1][0]**2)**0.5 < E:

        print([f'%.{6}f' % i for i in x_0], "is optimum\n")
        break
    H = Hessian(x_0[0][0], x_0[1][0])
    I = np.array([[1, 0], [0, 1]])

    res["a0"].append(f'%.{6}f' % x_0[0][0])
    res["a1"].append(f'%.{6}f' % x_0[1][0])
    res["g0"].append(f'%.{6}f' % g[0][0])
    res["g1"].append(f'%.{6}f' % g[1][0])
    res["delta"].append(f'%.{6}f' % delta)


    h = sol(H + delta*I, g)                  # solution of linear system of Hh
```

```
        x_1 = x_0 - h

        res["f0"].append(f'%.{6}f' % fv(x_0))
        res["f1"].append(f'%.{6}f' % fv(x_1))

        if fv(x_1) < fv(x_0):
            delta = c1 * delta
            x_0 = x_1
        else:
            delta = c2*delta
            x_0 = x_1

    return pd.DataFrame(res)

marquadrt_min()
```

```
x**2 + x*y + 2*y**2
['0.000020', '-0.000009'] is optimum
```

Out[24]:

	a0	a1	delta	g0	g1	f0	f1
0	10.000000	-10.000000	10000.000000	10.000000	-30.000000	200.000000	199.900048
1	9.999000	-9.997001	2500.000000	10.000999	-29.989004	199.900048	199.501070
2	9.994998	-9.985023	625.000000	10.004973	-29.945094	199.501070	197.918319
3	9.978965	-9.937390	156.250000	10.020540	-29.770595	197.918319	191.790854
4	9.914468	-9.751212	39.062500	10.077724	-29.090379	191.790854	170.140848
5	9.652444	-9.069588	9.765625	10.235300	-26.625910	170.140848	113.044976
6	8.611689	-7.059751	2.441406	10.163626	-19.627316	113.044976	36.014309
7	5.529518	-3.534203	0.610352	7.524832	-8.607295	36.014309	2.599630
8	1.605563	-0.816135	0.152588	2.394992	-1.658977	2.599630	0.019746
9	0.143834	-0.064626	0.038147	0.223042	-0.114671	0.019746	0.000011
10	0.003405	-0.001454	0.009537	0.005357	-0.002410	0.000011	0.000000

In [27]:

```
def bfgs_min():
    import sympy as sp
    import numpy as np
    import pandas as pd
    import math

    x, y = sp.symbols('x y')
    t = sp.symbols('t')
    e = math.e
    # f = e**(x-1) + e**(-y + 1) + (x - y)**2
    # f = x - y + 2*x**2 + 2*x*y + y**2
    f = x**2 + x*y + 2*y**2

    print(f)

    def gradient(x0, x1):
```

```python
        g = [[f.diff(x).evalf(subs = {x:x0, y:x1}), f.diff(y).evalf(subs = {x:x0, y
        return (np.array(g)).T

    def Hessian(x0, x1):
        g = [f.diff(x), f.diff(y)]

        H = [[g[0].diff(x), g[1].diff(x)],
             [g[0].diff(y), g[1].diff(y)]]

        Hv = [[H[0][0].evalf(subs = {x:x0, y:x1}), H[0][1].evalf(subs = {x:x0, y:x1
              [H[1][0].evalf(subs = {x:x0, y:x1}), H[1][1].evalf(subs = {x:x0, y:x1

        return np.array(Hv)

    def fv(val):
        evalute = f.evalf(subs = {x:val[0], y:val[1]})
        return evalute

    def sol(H_0, g_0):
        h1, h2 = sp.symbols(('h1 h2'))
        h = np.array([[h1], [h2]])
        Hh = np.dot(H_0, h)
        eq1 = sp.Eq(Hh[0][0], g_0[0][0])
        eq2 = sp.Eq(Hh[1][0], g_0[1][0])
        R = sp.solve((eq1, eq2), (h1, h2))
        h = [[R[h1]], [R[h2]]]
        return np.array(h)


    x_0 = np.array([[10, -10]]).T
    E = 0.001
    res = {"a0":[], "a1":[], "d0":[], "d1":[], "alpha":[], "g0":[], "g1":[]}

    g_0 = gradient(x_0[0][0], x_0[1][0])

    if (g_0[0][0]**2 + g_0[1][0]**2)**0.5 < E:
        print("optimum is ", x_0)
        return None

    H = Hessian(x_0[0][0], x_0[1][0])

    H_0 = np.array([[1, 0], [0, 1]])

    while True:

        d = - sol(H_0, g_0)                    # solution of linear system of Hh = g

        alpha = - np.dot(d.T, g_0) / np.dot(d.T, np.dot(H, d))

        res["a0"].append(f'%.{6}f' % x_0[0][0])
        res["a1"].append(f'%.{6}f' % x_0[1][0])
        res["alpha"].append(f'%.{6}f' % alpha)
        res["d0"].append(f'%.{6}f' % d[0][0])
        res["d1"].append(f'%.{6}f' % d[1][0])
        res["g0"].append(f'%.{6}f' % g_0[0][0])
        res["g1"].append(f'%.{6}f' % g_0[1][0])
```

```
        s = alpha * d

        x_1 = x_0 + s

        if (((x_1 - x_0)[0][0]**2 + (x_1 - x_0)[1][0]**2)**0.5 < E) or (((g_0)[0][0
            break
        else:
            g_1 = gradient(x_1[0][0], x_1[1][0])
            if (g_1[0][0]**2 + g_1[1][0]**2)**0.5 < E:

                break

            y0 = g_1 - g_0

            p = np.dot(H_0, s)
            q = np.dot(s.T, H_0.T)

            H_1 = H_0 + np.dot(y0, y0.T) / np.dot(y0.T, s) - np.dot(p, q)/np.dot(s.
            H_0 = H_1
            g_0 = g_1
            x_0 = x_1

    print(x_1, "is optimum\n")
    return pd.DataFrame(res)

bfgs_min()
```

```
x**2 + x*y + 2*y**2
[[0]
 [0]] is optimum
```

Out[27]:

	a0	a1	d0	d1	alpha	g0	g1
0	10.000000	-10.000000	-10.000000	30.000000	0.312500	10.000000	-30.000000
1	6.875000	-0.625000	-15.039062	1.367188	0.457143	13.125000	4.375000

In [ ]:

In [ ]: