ДЗ 1 Номер 1

1.Реализуйте метод дихотомии, метод золотого сечения и метод Фибоначчи.

2.Выберите произвольную несимметричную относительно некоторой вертикальной оси унимодальную функцию.

3.Сравните сходимость методов по времени и по числу итераций, необходимых для достижения заданной точности.

4.Проанализируйте результаты.

ссылка на оригинальный материал)) [https://github.com/amkatrutsa/MIPT-Opt/blob/master/12-NumMethods/Seminar12.ipynb (https://github.com/amkatrutsa/MIPT-Opt/blob/master/12-NumMethods/Seminar12.ipynb)](https://github.com/amkatrutsa/MIPT-Opt/blob/master/12-NumMethods/Seminar12.ipynb)

In [3]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import numpy as np
from tqdm.auto import tqdm
import scipy.optimize
import math
```

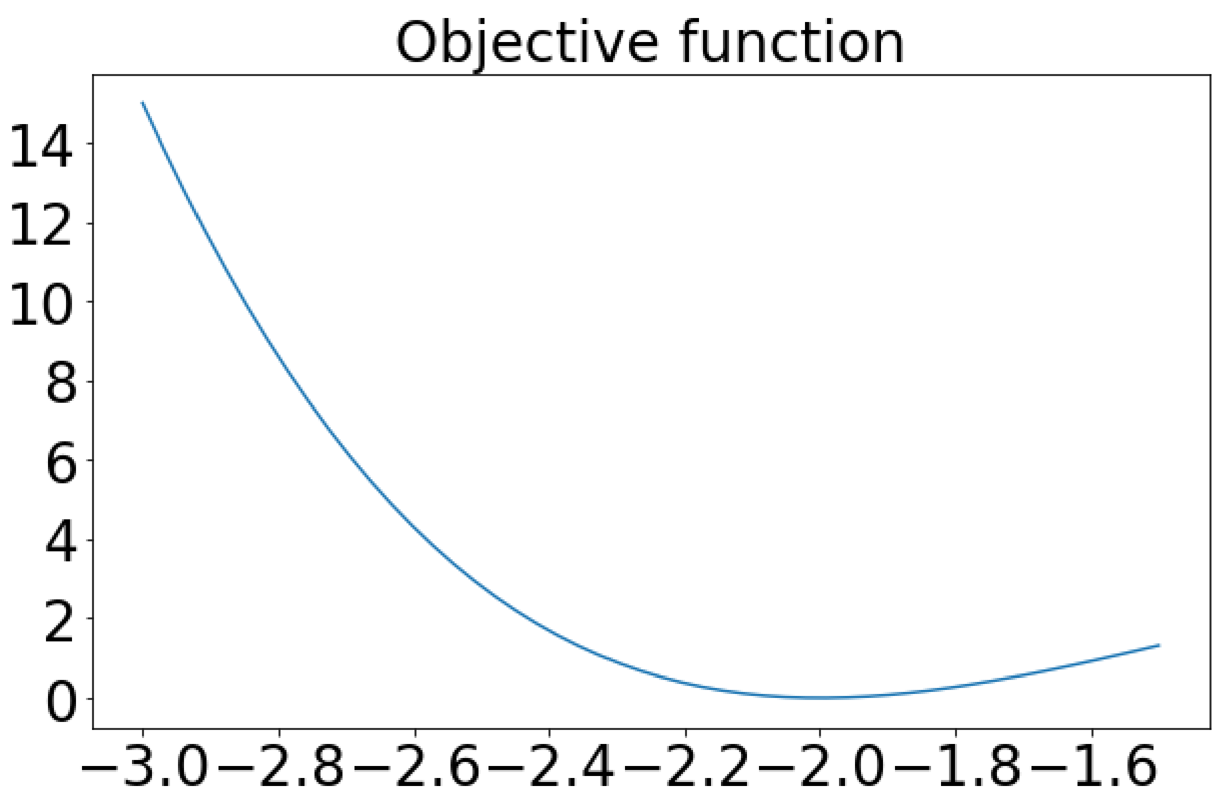
In [9]:
```
def binary_search(f, a, b, epsilon, callback=None):
    c = (a + b) / 2.0
    while abs(b - a) > epsilon:
#         Check left subsegment
        y = (a + c) / 2.0
        if f(y) <= f(c):
            b = c
            c = y
        else:
#         Check right subsegment
            z = (b + c) / 2.0
            if f(c) <= f(z):
                a = y
                b = z
            else:
                a = c
                c = z
        if callback is not None:
            callback(a, b)
    return c
```

In [10]:
```
def my_callback(a, b, left_bound, right_bound, approximation):
    left_bound.append(a)
    right_bound.append(b)
    approximation.append((a + b) / 2.0)
```

In [11]:

```python
left_boud_bs = []
right_bound_bs = []
approximation_bs = []

callback_bs = lambda a, b: my_callback(a, b,
            left_boud_bs,v right_bound_bs, approximation_bs)

# Target unimodal function on given segment
f = lambda x: (x - 2) * x * (x + 2)**2 # np.power(x+2, 2)
# f = lambda x: -np.sin(x)
x_true = -2
# x_true = np.pi / 2.0
a = -3
b = -1.5
epsilon = 1e-8
x_opt = binary_search(f, a, b, epsilon, callback_bs)
print(np.abs(x_opt - x_true))
plt.figure(figsize=(10,6))
plt.plot(np.linspace(a,b), f(np.linspace(a,b)))
plt.title("Objective function", fontsize=28)
plt.xticks(fontsize = 28)
_ = plt.yticks(fontsize = 28)
```

```
9.313225746154785e-10
```



In [ ]:

```python

```

ЗОЛОТОЕ СЕЧЕНИЕ

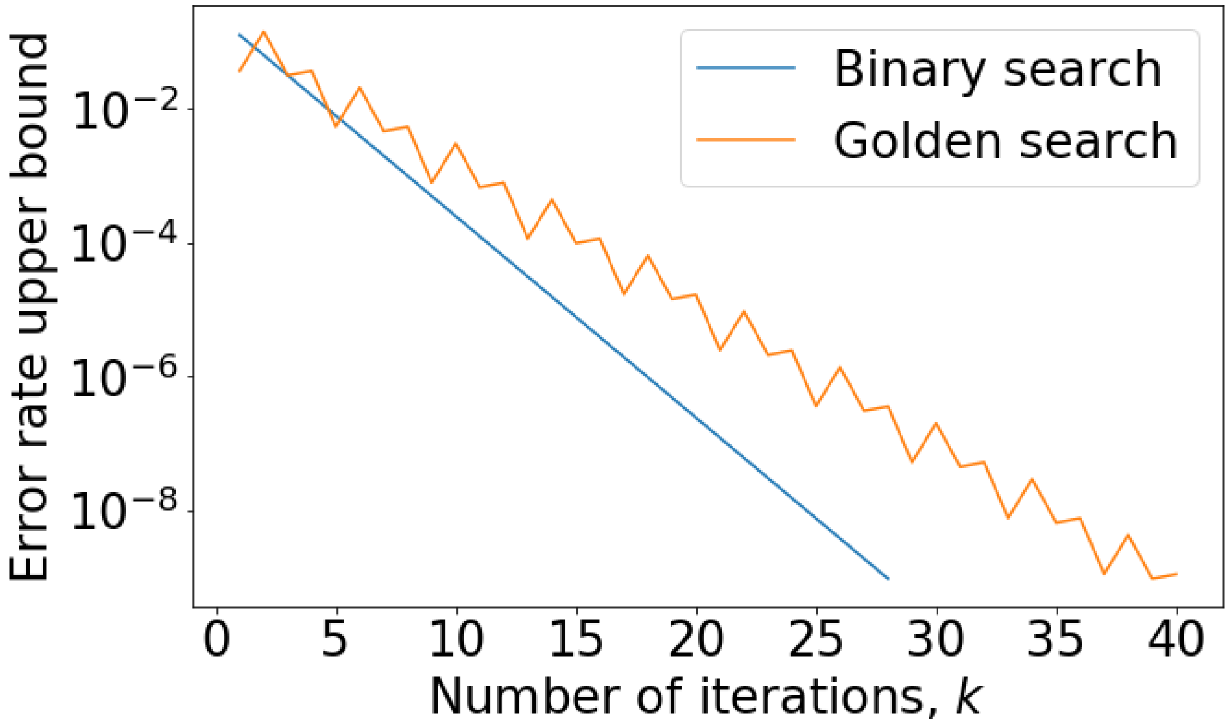
In [13]:
```
def golden_search(f, a, b, tol=1e-5, callback=None):
    tau = (np.sqrt(5) + 1) / 2.0
    y = a + (b - a) / tau**2
    z = a + (b - a) / tau
    while b - a > tol:
        if f(y) <= f(z):
            b = z
            z = y
            y = a + (b - a) / tau**2
        else:
            a = y
            y = z
            z = a + (b - a) / tau
        if callback is not None:
            callback(a, b)
    return (a + b) / 2.0
```

In [14]:
```
left_boud_gs = []
right_bound_gs = []
approximation_gs = []

cb_gs = lambda a, b: my_callback(a, b, left_boud_gs, right_bound_gs, app
x_gs = golden_search(f, a, b, epsilon, cb_gs)

print(f(x_opt))
print(f(x_gs))
print(np.abs(x_opt - x_true))
```

```
6.938893903875399e-18
9.549014390504221e-18
9.313225746154785e-10
```

Сравнение методов

In [15]:
```
plt.figure(figsize=(10,6))
plt.semilogy(np.arange(1, len(approximation_bs) + 1), np.abs(x_true - np
plt.semilogy(np.arange(1, len(approximation_gs) + 1), np.abs(x_true - np
plt.xlabel(r"Number of iterations, $k$", fontsize=26)
plt.ylabel("Error rate upper bound", fontsize=26)
plt.legend(loc="best", fontsize=26)
plt.xticks(fontsize = 26)
_ = plt.yticks(fontsize = 26)
```

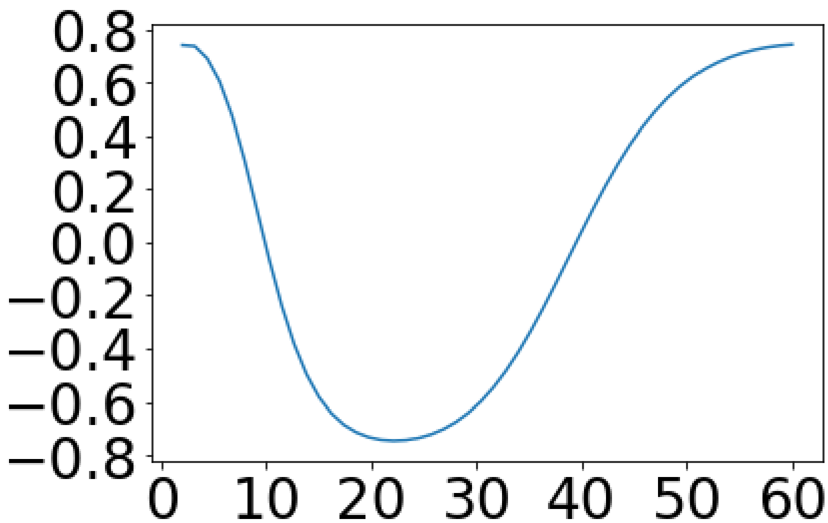


In [16]:
```
%timeit binary_search(f, a, b, epsilon)
%timeit golden_search(f, a, b, epsilon)
```

```
27.4 µs ± 3.49 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
177 µs ± 53.1 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

Пример иного поведения методов

In [17]:
```
f = lambda x: np.sin(np.sin(np.sin(np.sqrt(x))))
x_true = (3 * np.pi / 2)**2
a = 2
b = 60
epsilon = 1e-8
plt.plot(np.linspace(a,b), f(np.linspace(a,b)))
plt.xticks(fontsize = 28)
_ = plt.yticks(fontsize = 28)
```



Сравнение скорости сходимости и времени работы методов

Метод дихотомии

In [18]:
```
left_boud_bs = []
right_bound_bs = []
approximation_bs = []

callback_bs = lambda a, b: my_callback(a, b,
            left_boud_bs, right_bound_bs, approximation_bs)

x_opt = binary_search(f, a, b, epsilon, callback_bs)
print(np.abs(x_opt - x_true))
```

```
2.1968899233115735e-07
```

Метод золотого сечения

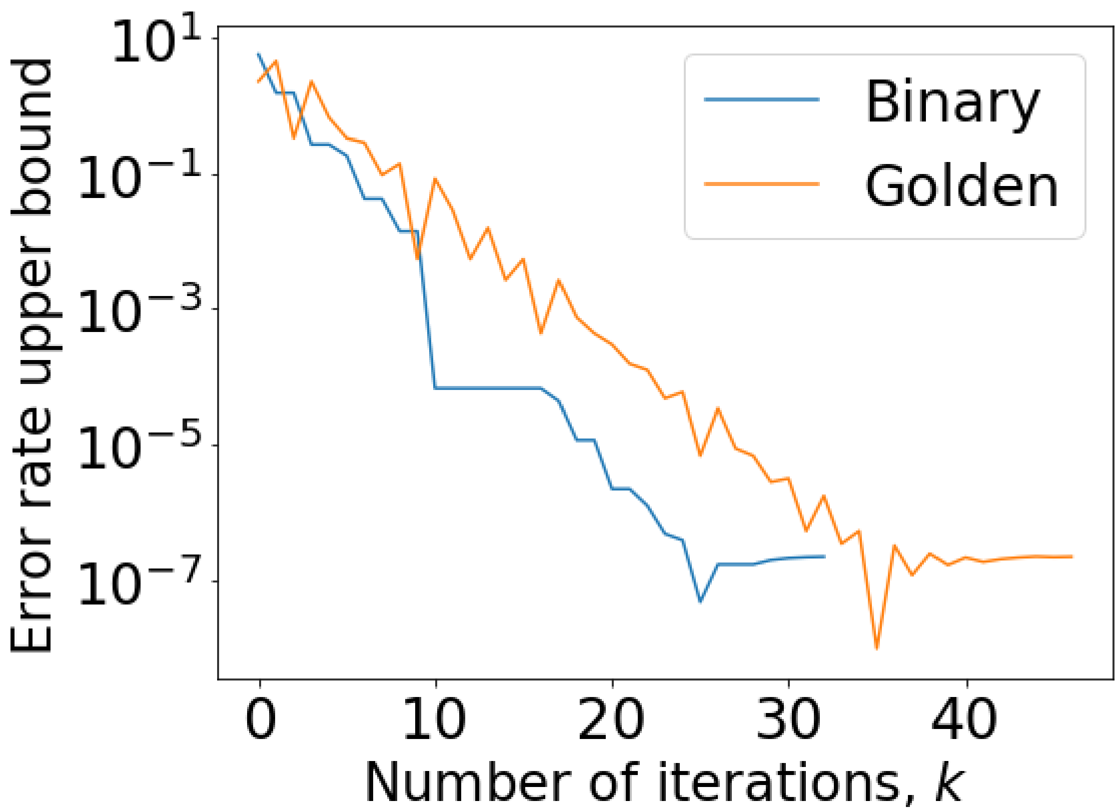
In [19]:
```
left_boud_gs = []
right_bound_gs = []
approximation_gs = []

cb_gs = lambda a, b: my_callback(a, b, left_boud_gs, right_bound_gs, app
x_gs = golden_search(f, a, b, epsilon, cb_gs)

print(np.abs(x_opt - x_true))
```

```
2.1968899233115735e-07
```

Сходимость

In [20]:
```
plt.figure(figsize=(8,6))
plt.semilogy(np.abs(x_true - np.array(approximation_bs, dtype=np.float64
plt.semilogy(np.abs(x_true - np.array(approximation_gs, dtype=np.float64
plt.legend(fontsize=28)
plt.xticks(fontsize=28)
_ = plt.yticks(fontsize=28)
plt.xlabel(r"Number of iterations, $k$", fontsize=26)
plt.ylabel("Error rate upper bound", fontsize=26)
```

Out[20]:
```
Text(0, 0.5, 'Error rate upper bound')
```



Время работы

In [21]:
```
%timeit binary_search(f, a, b, epsilon)
%timeit golden_search(f, a, b, epsilon)
```

```
1.14 ms ± 577 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
1.03 ms ± 487 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

In [4]:
```

math.e
```

Out[4]:
```
2.718281828459045
```

In [ ]:
```

```

Все последующие решения в оригинале предложены Елизаветой Козловой

Снова задание 1, но более понятным мне видом

In [ ]:
```

```

In [28]:
```
a=-0.45
b=0.7
def f(x):
    return (pow(math.e,-x*x)*(x*x+x))
```

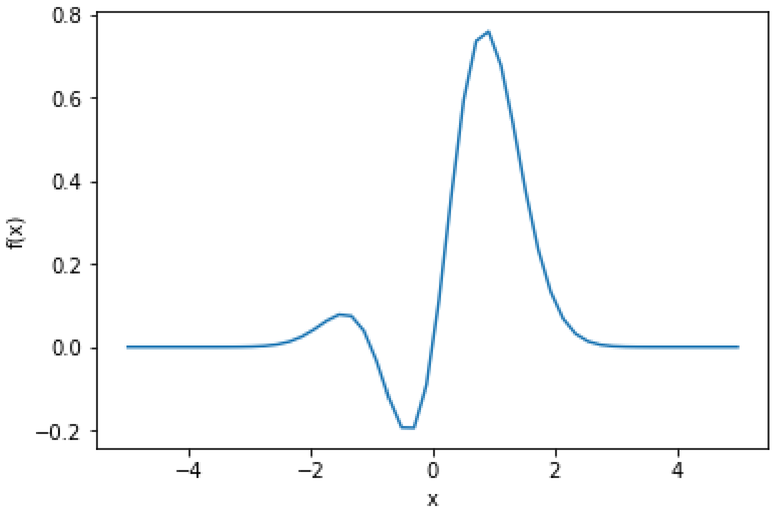
In [29]:
```
f(0)
```

Out[29]:
```
0.0
```

In [30]:
```
f(3)
```

Out[30]:
```
0.0014809176490401554
```

In [35]:
```python
x = np.linspace(-5, 5)
plt.plot(x, f(x))
plt.xlabel('x')
plt.ylabel('f(x)')
plt.show()
```



Судя по вольфраму (считать немного лень), минимум функции равен -0.20452 примерно в точке x=-0.40303

Зададим также критерий остановки. С учетом написанного выше, можно было выбрать и критерий остановки по значению функции или аргумента

In [31]:
```python
k_max = 100
```

In [101]:
```python
def dichotomy(f, a, b, k_max, write = None):
    x = (1/2)*(a+b)
    for k in range(k_max):
        y = (1/2)*(a+x)
        if f(y) <= f(x):
            b = x
            x = y
        else:
            z = (1/2)*(x+b)
            if f(x) <= f(z):
                a = y
                b = z
            else:
                a = x
                x = z
        if write is not None:
            write.append(f(x))
    return f(x)
```

In [102]:
```python
dichotomy(f, a, b, k_max)
```

Out[102]: -0.20452483320406528

По точности функции описано ниже. Пожалуй, стоит добавлять еще одно условие на значение аргумента или числа шагов, потому чо иттерационный метод может осциллировать в некоторых ситуациях

In [103]:
```
eps=0.0001
fex=-0.20452
```

In [104]:
```
def dichotomy1(f, a, b, k_max, fex, eps, write = None):
    x = (1/2)*(a+b)
    while (abs(f(x)-fex)>eps):
        y = (1/2)*(a+x)
        if f(y) <= f(x):
            b = x
            x = y
        else:
            z = (1/2)*(x+b)
            if f(x) <= f(z):
                a = y
                b = z
            else:
                a = x
                x = z
        if write is not None:
            write.append(f(x))
    return f(x)
```

In [105]:
```
dichotomy1(f, a, b, k_max,fex,eps)
```

Out[105]:
```
-0.2044706324449659
```

Метод золотого сечения

In [106]:
```
def get_y(a, b):
    return a + (2/(3 + np.sqrt(5)))*(b-a)

def get_z(a, b):
    return a + ((np.sqrt(5) - 1)/2)*(b-a)

def gold(f, a, b, k_max, write = None):
    y = get_y(a, b)
    z = get_z(a, b)
    for k in range(k_max):
        if f(y) <= f(z):
            b = z
            z = y
            y = get_y(a, b)
        else:
            a = y
            y = z
            z = get_z(a, b)
        if write is not None:
            write.append(f((a+b)/2))
    return f((a+b)/2)
```

In [107]:
```
gold(f, a, b, k_max)
```
Out[107]: -0.20452483320406525

In [ ]:
```

```

Метод Фибоначчи

In [108]:
```
fibonacci = [0, 1]
for i in range(2, 20):
    fibonacci.append(fibonacci[i-2] + fibonacci[i-1])
```

In [109]:
```
def get_l(k_max, k, a, b):
    return a + (fibonacci[k_max - k - 1]/fibonacci[k_max - k + 1])*(b-a)

def get_m(k_max, k, a, b):
    return a + (fibonacci[k_max - k]/fibonacci[k_max - k + 1])*(b-a)

def fib(f, a, b, k_max, write = None):
    k_max += 1
    l = get_l(k_max, 1, a, b)
    m = get_m(k_max, 1, a, b)
    for k in range(2, k_max + 1):
        if f(l) <= f(m):
            b = m
            m = l
            l = get_l(k_max, k, a, b)
        else:
            a = l
            l = m
            m = get_m(k_max, k, a, b)
        if write is not None:
            write.append(f((a+b)/2))

    x = (a+b)/2
    return f(x)
```

In [110]:
```
fib(f, a, b, 10)
```
Out[110]: -0.20452142905152423

In [111]:
```
x=-0.4030317
f_exact=f(x)
```

In [112]:
```
fib_res = []
gold_res = []
dich_res = []
```

In [113]:
```
%'.format(np.abs((f_exact - fib(f, a, b, k_max, write = fib_res))/f_exact)))
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 5.96 µs

---------------------------------------------------------------------------
IndexError                                Traceback (most recent call las
t)
<ipython-input-113-eff6a3635e4a> in <module>
      1 get_ipython().run_line_magic('time', '')
----> 2 print('Ошибка Фибоначчи {:.2e}%'.format(np.abs((f_exact - fib(f,
 a, b, k_max, write = fib_res))/f_exact)))

<ipython-input-109-1c20dafe8431> in fib(f, a, b, k_max, write)
      7 def fib(f, a, b, k_max, write = None):
      8     k_max += 1
----> 9     l = get_l(k_max, 1, a, b)
     10     m = get_m(k_max, 1, a, b)
     11     for k in range(2, k_max + 1):

<ipython-input-109-1c20dafe8431> in get_l(k_max, k, a, b)
      1 def get_l(k_max, k, a, b):
----> 2     return a + (fibonacci[k_max - k - 1]/fibonacci[k_max - k + 1]
)*(b-a)
      3
      4 def get_m(k_max, k, a, b):
      5     return a + (fibonacci[k_max - k]/fibonacci[k_max - k + 1])*(b
-a)

IndexError: list index out of range
```

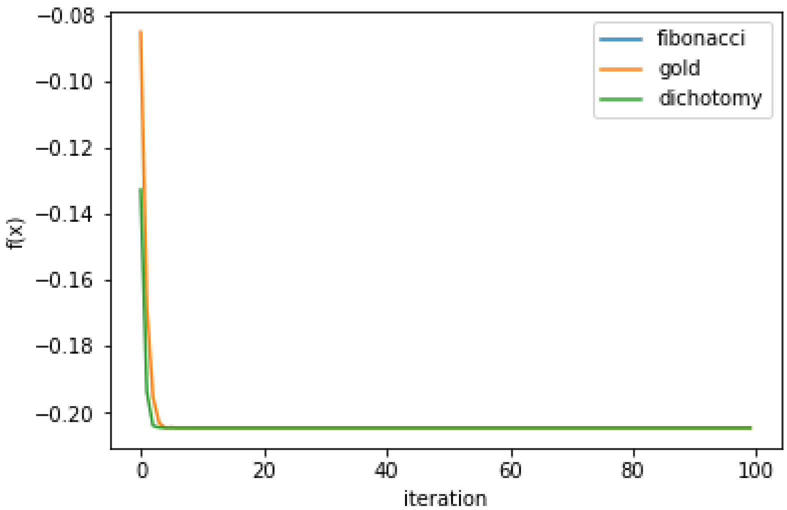
In [114]:
```
.format(np.abs((gold(f, a, b, k_max, write = gold_res) - f_exact)/f_exact)))
```

```
CPU times: user 2 µs, sys: 1 µs, total: 3 µs
Wall time: 5.25 µs
Ошибка золотого сечения 1.49e-15%
```

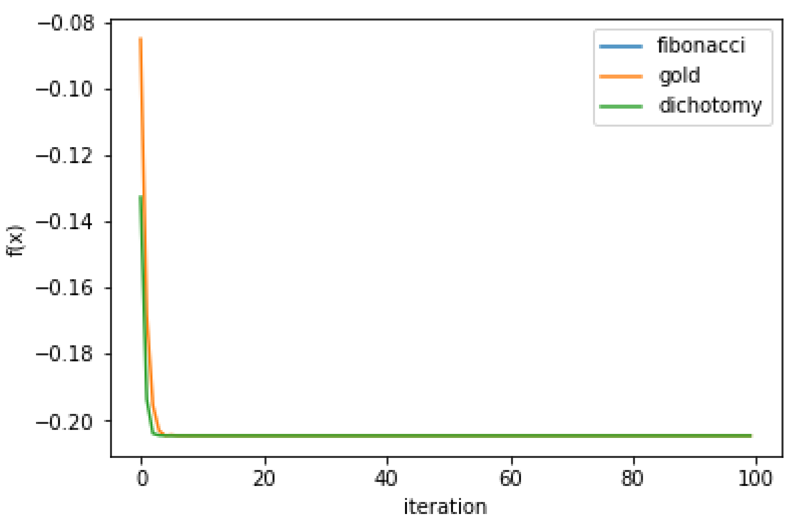
In [115]:
```
at(np.abs((dichotomy(f, a, b, k_max, write = dich_res) - f_exact)/f_exact)))
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 4.77 µs
Ошибка дихотомии 1.63e-15%
```

In [116]:
```
plt.plot(fib_res)
plt.plot(gold_res)
plt.plot(dich_res)
plt.legend(['fibonacci', 'gold', 'dichotomy'])
plt.xlabel('iteration')
plt.ylabel('f(x)')
plt.show()
```



In [117]:
```
plt.plot(fib_res)
plt.plot(gold_res)
plt.plot(dich_res)
plt.legend(['fibonacci', 'gold', 'dichotomy'])
plt.xlabel('iteration')
plt.ylabel('f(x)')
plt.show()
```



In [ ]:
```
чи какие-то проблемы возникли при реализации, нужно время, чтобы разобраться
```

In [ ]:
```
def grad_f(x):
    return np.sinh(x)/np.cosh(x)
def hess_f(x):
    return 1 - grad_f(x)**2
```

Номер 6

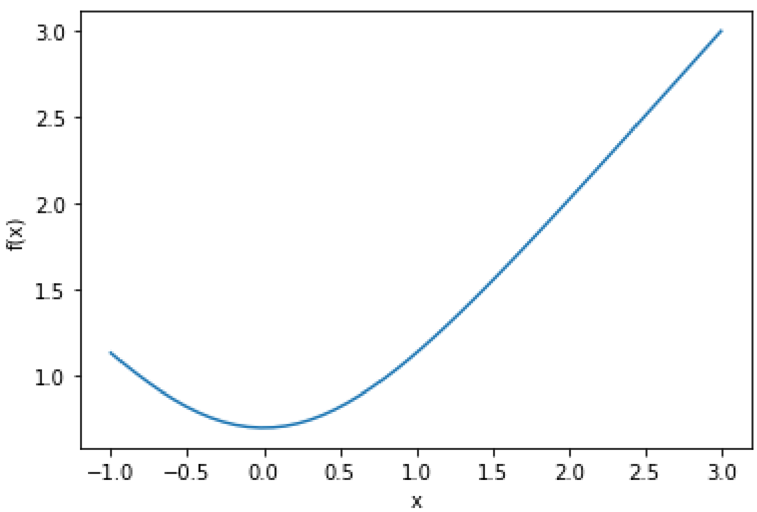
In [118]:
```
def f(x):
    return np.log(np.exp(x) + np.exp(-x))

```

In [121]:
```
def grad_f(x):
    return (pow(math.e,2*x)-1)/(pow(math.e,2*x)+1)
```

In [122]:
```
def hess_f(x):
    return (4*pow(math.e,2*x))/(pow(math.e,2*x)+1)**2
```

In [123]:
```
def next_x(x):
    return x - grad_f(x)/hess_f(x)
```

In [124]:
```
x = np.linspace(-1, 3)
plt.plot(x, f(x))
plt.xlabel('x')
plt.ylabel('f(x)')
plt.show()
```



In [126]:
```
k = 10
```

In [127]:
```
x = 1
x_1 = [x]
for i in range(k):
    x = next_x(x)
    x_1.append(x)
    print(x)
```

```
-0.8134302039235091
0.40940231658338533
-0.047304916455615464
7.060280364457744e-05
-2.346337642407381e-13
1.187201384374721e-17
1.187201384374721e-17
1.187201384374721e-17
1.187201384374721e-17
1.187201384374721e-17
```

In [132]:
```
x = 1.1
x_11 = [x]
for i in range(int(k/2)):
    x = next_x(x)
    x_11.append(x)
    print(x)
```

```
-1.1285525852679466
1.2341311330390985
-1.6951659799227912
5.715360100379576
-23021.35648572018
```

In [ ]:
```

```

Демпфированный метод Ньютона (с переменным шагом)

можно было найти альфу аналитически (одномерный поиск),

$$\alpha_k = \arg\min_{\alpha>0} f\left(x_k - \alpha\left[\nabla^2 f(x_k)\right]^{-1}\nabla f(x_k)\right)$$

тогда метод сошелся бы за 1 шаг, что показано письменно на листочке

а можно исползовать дробление, начиная

$$\alpha = 1, \alpha_k = \gamma\alpha_k, 0 < \gamma < 1$$

до выполнения каких-то условий, например

$$f(x_{k+1}) \leq f(x_k) - \alpha q\left(\left[\nabla^2 f(x_k)\right]^{-1}\nabla f(x_k), \nabla f(x_k)\right)$$

$$\|\nabla f(x_{k+1})\|^2 \leq (1-\alpha q)\|\nabla f(x_k)\|^2$$

$$0 < q < 1$$

In [135]:
```
def d_next_x(x, a):
    a *= 1/2
    x = x - a*grad_f(x)/hess_f(x)
    return x, a
```

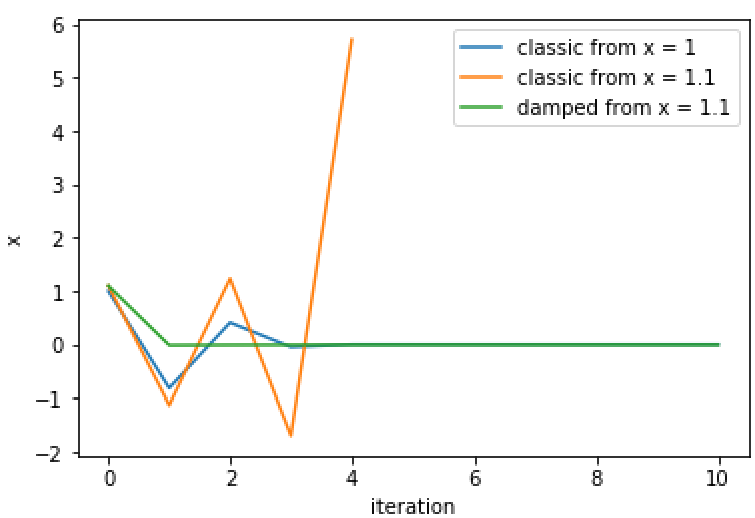
In [136]:
```
x = 1.1
a = 1
x_d = [x]
for i in range(k):
    x, a = d_next_x(x, a)
    x_d.append(x)
    print(x)
```

```
-0.014276292633973231
-0.010706734507821794
-0.00936829041220263
-0.008782738002224217
-0.008508263325487925
-0.008375315295123687
-0.008309880084472596
-0.008277418121019516
-0.008261250550287758
-0.008253182555724535
```

Теперь все стало хорошо

Графики

In [139]:
```
plt.plot(x_1)
plt.plot(x_11[:-1])
plt.plot(x_d)
plt.legend(['classic from x = 1', 'classic from x = 1.1', 'damped from x = 1.
plt.xlabel('iteration')
plt.ylabel('x')
plt.show()
```




```
 
```

В общем, тут очень хорош демпфированный метод Ньютона

f

Задача 7

In [217]:
```
k = 400
```

In [218]:
```
def f(x):
    x1, x2 = x
    return 100*(x2-x1**2)**2 + (1-x1)**2
def grad_f(x):
    x1, x2 = x
    return np.array([-2*x1*200*(x2-x1**2) - 2*(1-x1), 200*(x2-x1**2)])
def hess_f(x):
    x1, x2 = x
    a11 = 1200*x1**2 + 2
    a12 = -400*x1
    a21 = -400*x1
    a22 = 200
    return np.array([[a11, a12], [a21, a22]])
```

Армихо

In [219]:
```
def get_a(d):
    a = 0.9
    t = 0.9
    eps = 0.5
    while True:
        f1 = f(x - a*d)
        f2 = f(x) - eps*a*np.dot(d, d)
        if f1 < f2:
            return a
        else:
            a *= t
```

In [220]:
```
def next_x(x, a):
    return x - a*grad_f(x)
```

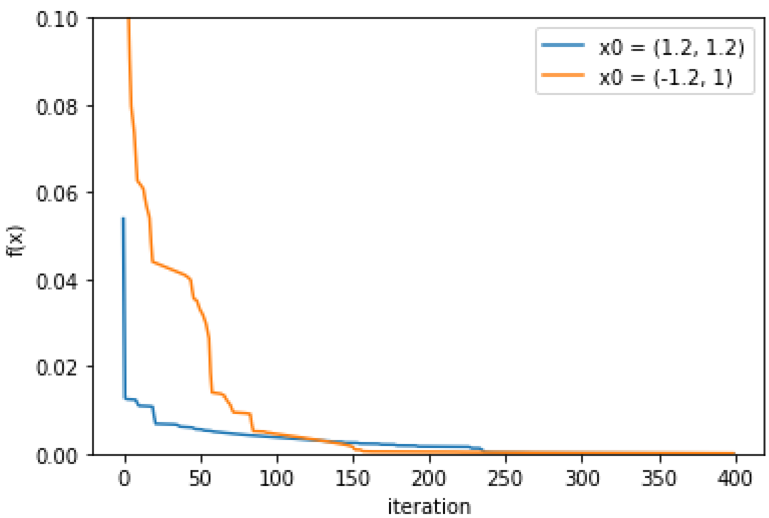
In [221]:
```
f_1 = []
a_1 = []
x = np.array([1.2, 1.2])
for i in range(k):
    a = get_a(grad_f(x))
    x = next_x(x, a)
    f_1.append(f(x))
    a_1.append(a)
```

In [222]:

```
f_2 = []
a_2 = []
x = np.array([-1.2, 1])
for i in range(k):
    a = get_a(grad_f(x))
    x = next_x(x, a)
    f_2.append(f(x))
    a_2.append(a)
```

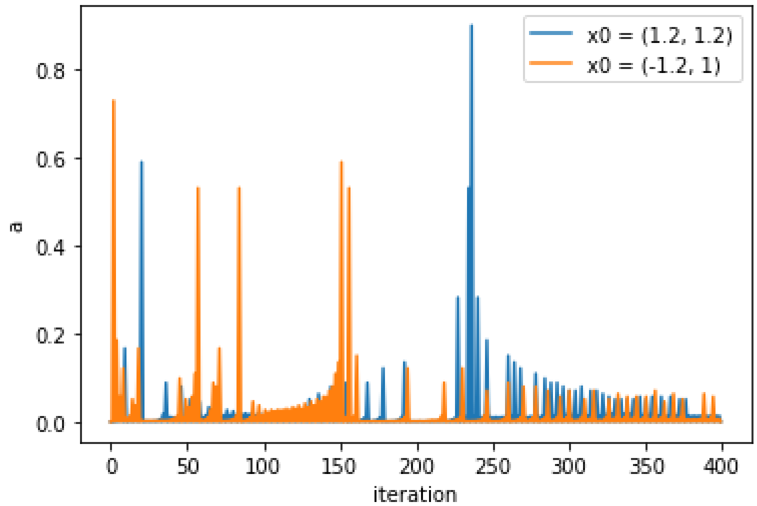
In [223]:

```
plt.plot(f_1)
plt.plot(f_2)
plt.legend(['x0 = (1.2, 1.2)', 'x0 = (-1.2, 1)'])
plt.xlabel('iteration')
plt.ylabel('f(x)')
plt.ylim(0, 0.1)
plt.show()
```



Сходимость в зависимости от нач условий

In [224]:
```
plt.plot(a_1)
plt.plot(a_2)
plt.legend(['x0 = (1.2, 1.2)', 'x0 = (-1.2, 1)'])
plt.xlabel('iteration')
plt.ylabel('a')
plt.show()
```



Длина шага резко меняется

In [225]:
```
k = 400
x0 = np.array([-1.2, 1])
def next_B(B, s, y):
    return (np.eye(2) - np.outer(s, y)/(np.dot(y, s))).dot(B).dot((np.eye
def get_alpha(x, B = np.eye(2)):
    return scipy.optimize.minimize(lambda a: f(x - a*B.dot(grad_f(x))), x
```

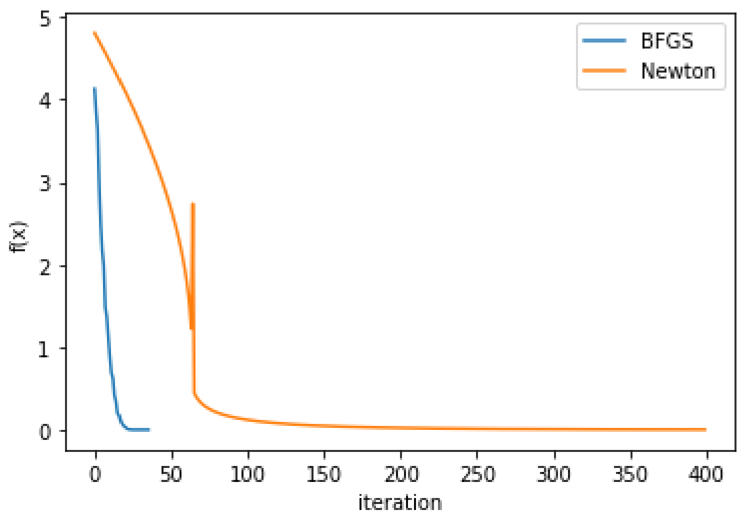
In [226]:
```
f_bfgs = []
B = np.eye(2)
x = x0
gr_f = grad_f(x)
for i in range(k):
    alpha = get_alpha(x, B)
    x_new = x - alpha*B.dot(gr_f)
    gr_f_new = grad_f(x_new)
    y = gr_f_new - gr_f
    s = x_new - x
    B = next_B(B, s, y)
    x = x_new
    gr_f = gr_f_new
    f_bfgs.append(f(x_new))
```

```
/Users/dmitrii/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launch
er.py:4: RuntimeWarning: invalid value encountered in true_divide
  after removing the cwd from sys.path.
```

In [227]:
```
def newton(x):
    return x - np.linalg.inv(hess_f(x)).dot(grad_f(x))
```

In [228]:
```
f_newton = []
x = x0
for i in range(k):
    x = newton(x)
    f_newton.append(f(x))
```

In [229]:
```
plt.plot(f_bfgs)
plt.plot(f_newton)
plt.legend(['BFGS', 'Newton'])
plt.xlabel('iteration')
plt.ylabel('f(x)')
# plt.ylim(0, 0.1)
plt.show()
```



Происходит что-то странное, возможно, из-за проблем в коде, но методы сходятся к нулю (БФГШ точно, причем намного быстре)

In [ ]:

In [ ]:

In [ ]:

In [ ]: