Homework1

18 февраля 2018 г.

1 Методы оптимизации, ДЗ 1

1.1 Сазанович Владислав M3439, Вариант 110

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

Для начала зададим функцию $f(x) = exp(\sqrt{x}) + 11 \cdot exp(-11x)$ и желаемую погрешость.

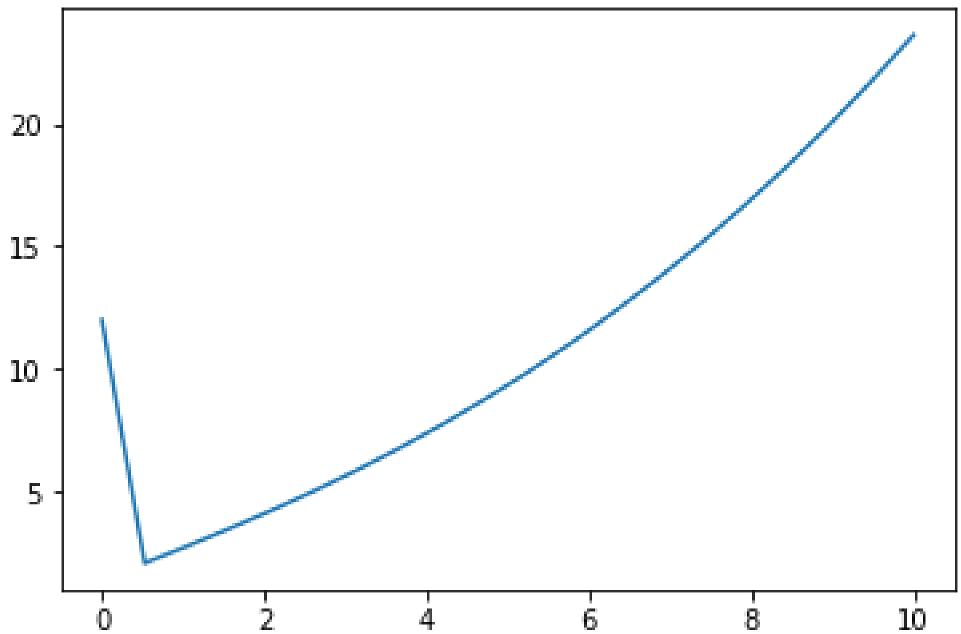
In [2]:
```
eps = 1e-6

def f(x):
    return np.exp(np.sqrt(x)) + 11 * np.exp(-11 * x)
```

1.1.1 Сканирование

Разбиваем отрезок [0..10] на 20 частей и считаем значение функции в них.

In [3]:
```
x = np.linspace(0, 10, 20)
y = f(x)
plt.plot(x, y)
plt.show()
```



Можно сказать, что минимум будет точно находится на отрезке [0..2]

```
In [4]: LEFT = 0.
        RIGHT = 2.
```

1.2 Метод дихотомии

```
In [5]: def dichotomy(f):
            l = LEFT
            r = RIGHT

            while (np.abs(r - l) / 2 > eps):
                # находим две центральные точки
                delta = (r - l) / 6
                m1 = (l + r) / 2 - delta
                m2 = (l + r) / 2 + delta

                # обновляем границы
                if f(m1) <= f(m2):
                    r = m2
                else:
                    l = m1

            return l, r
```

1.3 Метод золотого сечения

In [6]:
```
def golden_ratio(f):
    l = LEFT
    r = RIGHT

    # Значения функции в промежуточных точках неизвестны
    f1 = None
    f2 = None

    while (np.abs(r - l) / 2 > eps):
        m1 = l + (3 - np.sqrt(5)) / 2 * (r - l)
        m2 = l + (np.sqrt(5) - 1) / 2 * (r - l)

        # Вычисляем f если нет значения с предыдущего шага
        if f1 is None:
            f1 = f(m1)

        # Вычисляем f если нет значения с предыдущего шага
        if f2 is None:
            f2 = f(m2)

        if f1 <= f2:
            r = m2
            f2 = f1
            f1 = None
        else:
            l = m1
            f1 = f2
            f2 = None

    return l, r
```

1.4 Метод Фибоначчи

In [7]:
```
def fibonacci(f):
    l = LEFT
    r = RIGHT

    # Считаем числа Фибоначчи
    fib = [1, 1]
    while (fib[-1] < (r - l) / eps):
        fib.append(fib[-1] + fib[-2])
    n = len(fib) - 1

    # Считаем начальное приближение
    m1 = l + (r - l) * (fib[n - 2] / fib[n])
    f1 = f(m1)
```

```
        m2 = l + (r - l) * (fib[n - 1] / fib[n])
        f2 = f(m2)

        for k in range(1, n - 1):
            # Вычисляем f если нет значения с предыдущего шага
            if f1 is None:
                f1 = f(m1)

            # Вычисляем f если нет значения с предыдущего шага
            if f2 is None:
                f2 = f(m2)

            if f1 <= f2:
                r = m2
                m2 = m1
                m1 = l + (r - l) * (fib[n - k - 2] / fib[n - k])
                f2 = f1
                f1 = None
            else:
                l = m1
                m1 = m2
                m2 = l + (r - l) * (fib[n - k - 1] / fib[n - k])
                f1 = f2
                f2 = None

        return l, r
```

1.5 Анализ производительности

1.5.1 Парочка вспомогательных функций

```
In [8]: # Функция которая оборачивает f для подсчета количества ее вызовов.
        def perf(function, invocations):
            # Добавляет 1 к счетчику и вызывает function
            def invoke_f(x, invocations):
                invocations[0] += 1
                return function(x)

            # Возвращаем фунцкию f которая будет дополнительно считать количество вызовов
            return lambda x: invoke_f(x, invocations)
```

```
In [9]: # Функция которая запускает алгоритм и пишет результат красиво
        def measure_function(algo, algo_name):
            invocations = [0]
            l, r = algo(perf(f, invocations))

            print("{}:\n    Результат: [{} .. {}] (delta = {}), Количество вызовов: {}".format(a
```

1.5.2 Анализ алгоритмов

```
In [10]: measure_function(dichotomy, "Метод дихотомии")
```

```
Метод дихотомии:
    Результат: [0.3998413345283686 .. 0.3998427080513449] (delta = 1.3735229763001122e-06), Коли
```

```
In [11]: measure_function(golden_ratio, "Метод золотого сечения")
```

```
Метод золотого сечения:
    Результат: [0.39984134329275545 .. 0.3998430826485503] (delta = 1.7393557948386373e-06), Кол
```

```
In [12]: measure_function(fibonacci, "Метод Фибоначчи")
```

```
Метод Фибоначчи:
    Результат: [0.3998413448229796 .. 0.3998431811097507] (delta = 1.8362867710841613e-06), Коли
```