Решение систем нелинейных уравнений методом секущих.

Описание метода.

Дана система из n нелинейных уравнений с n неизвестными. Они будут иметь вид:

$$\mathbf{f}(\mathbf{x}) = \mathbf{0} \Leftrightarrow \begin{cases} f_1(x_1, \ldots, x_s) = 0 \\ \cdots \quad \cdots \quad \cdots \\ f_s(x_1, \ldots, x_s) = 0 \end{cases}$$

Будем итеративно искать решение с помощью метода секущих. Пусть x_0 начальное приближение, x_k k-е приближение. В основе метода лежит разложение функции в ряд Тейлора векторной функции $f(x)$.

$$f(x) \approx f(x_k) + f'(x_k)(x - x_k)$$

После преобразований получаем выражение для вычисления следующего приближения.

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{f}_k'^{-1}\mathbf{f}_k$$

Матрица $f'(x)$ будет иметь вид:

$$\mathbf{f}_k' = \mathbf{f}'(\mathbf{x}_k) \approx \mathbf{H}_k^{-1}\mathbf{\Gamma}_k$$

Где **H** и **Г** будут на каждом k-м шаге равны:

$$\mathbf{H}_k = h\mathbf{I},\ \mathbf{\Gamma}_k = \left\{\gamma_{ij}^{(k)}\right\}_{i,j}^{s},$$
$$\gamma_{ij}^{(k)} = f_i(\mathbf{x}_k + h\mathbf{e}_j) - f_i(\mathbf{x}_k)$$

Тогда с помощью этих формул будут находиться приближение вектора решений по мере итераций.

Также о данном методе можно сказать, что он имеет квадратичную сходимость.

Программная реализация

```python
import numpy as np

h = 10E-6


def get_h(n):
    res = np.zeros([n, n])
    for i in range(n):
        res[i, i] = h
    return res


def get_g(f, x0):
    n = len(f)
    res = np.zeros([n, n])
    for i in range(n):
        for j in range(n):
            x = np.zeros(n)
            for k in range(n):
                x[k] = x0[k]
            x[j] += h
            res[i, j] = f[i](x) - f[i](x0)
    return res
```

```python
def nsolve(f, it):
    n = len(f)
    x = np.zeros(n)
    hm = get_h(n)
    for i in range(it):
        g = get_g(f, x)
        fdk = np.linalg.inv(g).dot(hm)
        fk = np.zeros(n)
        for j in range(n):
            fk[j] = f[j](x)
        x -= fdk.dot(fk)
        if np.linalg.norm(fk)<10E-12:
            print('Нужная точность была достигнута')
            print('Кол-во итераций: ', i)
            break
    return x


f1 = [(lambda x: np.sin(x[0]) + x[1]),
      (lambda x: x[1] ** 2 - 10 * x[0] + 2)]

f2 = [(lambda x: x[1]**1.2-np.sqrt(x[0]*10+2)+0.1),
      (lambda x: 3-x[0] ** 1.7 + np.exp(-x[1]/10+2))]

print(nsolve(f1, 10))
print()
print(nsolve(f2, 10))
```

Проверка правильности решения

Возьму две функции и посмотрю, что для них получится

$$f_1(x,y) = \begin{pmatrix} \sin(x) + y \\ y^2 - 10x + 2 \end{pmatrix}$$

$$f_2(x,y) = \begin{pmatrix} y^{1.2} - \sqrt{10x+2} + 0.1 \\ 3 - x^{1.7} + e^{-\frac{y}{10}+2} \end{pmatrix}$$

Результат работы программы для этих функций:

```
Нужная точность была достигнута
Кол-во итераций:  3
[ 0.2041085  -0.20269424]

Нужная точность была достигнута
Кол-во итераций:  8
[3.34233049 4.3591788 ]
```

Точность была выставлена как 10^{-12}. Программа останавливается, когда норма вектора невязки становится чем выбранное число точности. Таким образом норма вектора невязки была близка к нулю с точностью до 12 знаков после запятой.

Здесь значения были вычислены меньше, чем за 10 итераций, и это подтверждает теоретические данные о скорости сходимости данного метода.

Исходя из этого, можно сказать, что программная реализация работает правильно и дает достаточно точный результат за небольшое количество шагов.