МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

(ФГБОУ ВО «ВГУ»)

Факультет прикладной математики, информатики и механики

Дисциплина: «Численные методы»

Лабораторное задание № 3

Отчет по лабораторной работе № 3

Тема: «Численные методы решения спектральных задач линейной алгебры»

Выполнил:

студент 3 курса 8 группы

Казанин Андрей Алексеевич

(*фио*)

Проверил:

преподаватель

Махинова О.А.

1. **Постановка задачи**

Методом прямых итераций с исчерпыванием определить третье по модулю максимальное собственное значение симметричной матрицы простой структуры и соответствующий ему собственный вектор.

**Для решения задачи процедуре подается:**

– размерность матрицы .

– максимальное собственное значение по модулю матрицы .

– соответствующий ему собственный вектор.

– второе максимальное собственное значение по модулю матрицы .

– соответствующий ему собственный вектор.

– точность определения второго максимального по модулю собственного значения.

– точность определения второго собственного вектора.

– максимально допустимое число итераций.

**Результатом работы процедуры являются:**

– второе максимальное по модулю собственное значение.

– второй собственный вектор.

– число выполненных итераций.

– мера точности полученной пары ().

1. **Метод решения**

Для начала рассмотрим алгоритм генерации матрицы с заранее известными собственными значениями и собственными векторами.

Пусть – диагональная матрица, где на главной диагонали стоят элементы вектора , содержащего собственные значения будущей матрицы.

Рассмотрим – случайно сгенерированный и нормированный () вектор.

Образуем с помощью вектора-столбца матрицу Хаусхолдера:

Теперь можно образовать тестирующую матрицу:

У этой матрицы известны все собственные значения (элементы вектора ) и собственные векторы (векторы-столбцы матрицы )

Теперь, рассмотрим алгоритм исчерпывания для нахождения третьего максимального по модулю собственного значения матрицы и соответствующего ему собственного вектора.

В дальнейшем полагаем:

Считаем пары и известными и поступившими на вход алгоритма.

Образуем матрицы:

Выберем начальное приближение

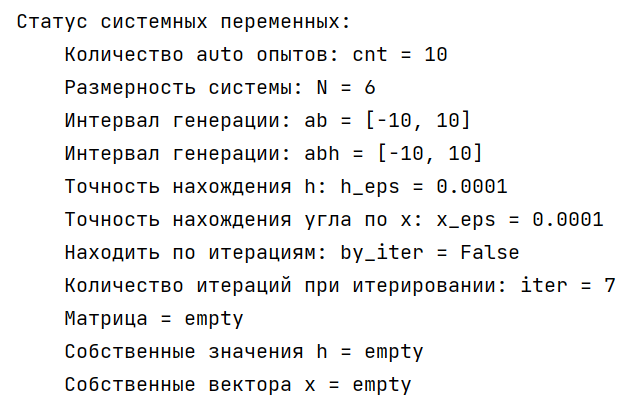
Организуем итерационный процесс по схеме:

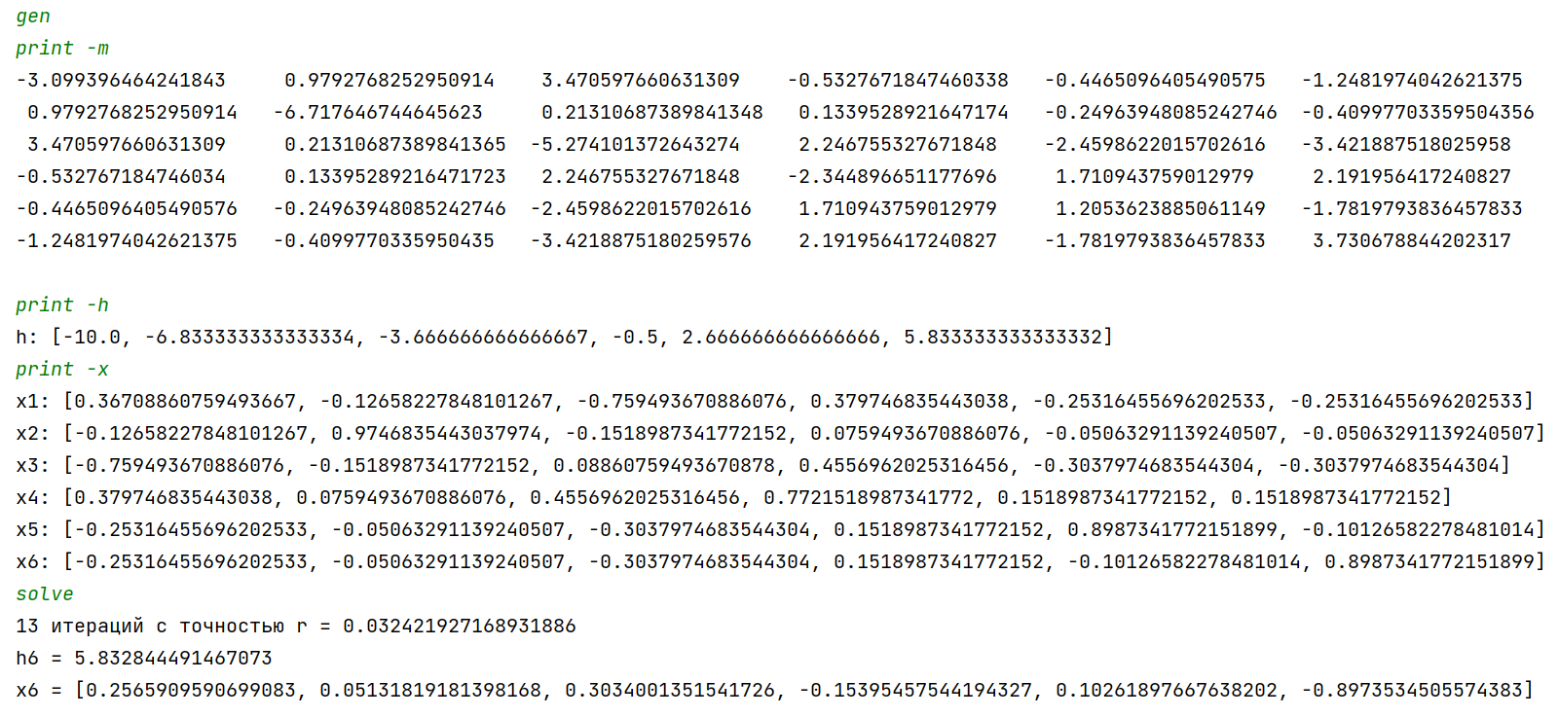
, k=0,1,2…

При этом:

, при

1. **Тестовый пример**

Сгенерируем матрицу по следующим настройкам:

В результате получаем:

1. **Вычислительные эксперименты**

Проведем серию вычислительных экспериментов с целью выявить основные закономерности касаемо таких параметров как: точность ε, средняя оценка точности собственных значений λ, средняя оценка точности собственных векторов , средняя мера точности и среднего числа операций, а также с целью выявления зависимостей между ними.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № теста | Размерность системы | Диапазон значений | Точность  () | Средняя оценка точности собственных значений λ | Средняя оценка точности собственных векторов | Средняя мера точности | Среднее число итераций |
| 1 | 10 | [-2; 2] | 10-5 | 1.3859237 | 0.9443101 | 0.0011721 | 21.1 |
| 2 | 10-8 | 1.3002985 | 0.5097729 | 3.9091711 | 34.7 |
| 3 | [-50; 50] | 10-5 | 0.0003403 | 1.1703400 | 0.0949194 | 25.7 |
| 4 | 10-8 | 3.8220534 | 0.7917322 | 0.0033978 | 36.9 |
| 5 | 30 | [-2; 2] | 10-5 | 7.0095820 | 1.1493653 | 0.0022882 | 61.3 |
| 6 | 10-8 | 7.7849103 | 0.9159111 | 7.9064988 | 115.6 |
| 7 | [-50; 50] | 10-5 | 0.0004504 | 0.7514979 | 0.1830902 | 116.6 |
| 8 | 10-8 | 4.4717525 | 0.9819963 | 0.0053722 | 196.1 |
| 9 | 50 | [-2; 2] | 10-5 | 0.0001412 | 1.5428407 | 0.0026434 | 97.0 |
| 10 | 10-8 | 1.4459448 | 1.1700913 | 8.7657785 | 191.7 |
| 11 | [-50; 50] | 10-5 | 0.0004644 | 0.7716903 | 0.1980741 | 283.1 |
| 12 | 10-8 | 4.6552464 | 0.7586531 | 0.0060565 | 449.5 |

1. **Выводы**

Из вычислительных экспериментов можно определить следующие зависимости:

* Среднее число итераций прямо пропорционально как росту точности, так и росту диапазона значений
* Средняя мера точности улучшается при росте точности на достаточном интервале генерации собственных значений
* Средняя оценка точности собственных векторов в целом улучшается при росте точности
* Средняя оценка точности собственных значений λ ухудшается при повышении точности, что связано с лишними итерациями, т.к. для алгоритма важна точность по векторам, которая достигается позднее, нежели на собственных значения.
* Недостаточность размаха интервала генерации собственных значений негативно влияет на точность собственных значений.

1. **Приложение**

Ниже приведены основные методы и функции, используемые приложением для проведения вычислительных экспериментов:

1. **Модуль Matrix:**

**from** \_\_future\_\_ **import** annotations  
  
**"""  
Модуль Matrix.py.  
Определяет базовую функциональность для реализации поставленной задачи.  
"""  
  
import** random  
**from** BaseFunctions **import** \*  
**from** typing **import** List  
  
  
**class** Matrix:  
 *"""  
 Класс квадратной матрицы.  
  
 Поля:  
 n - размерность матрицы.  
 a - буфер соответствующего этой матрице вложенного списка.  
 """* **def** \_\_init\_\_(self, n: int):  
 *"""  
 Инициализатор матрицы.  
 Создает матрицу размера NxN, заполненную 0.  
  
 Args:  
 n: размер создаваемой матрицы.  
 """* self.n = n  
 self.a = []  
 self.clear(0)  
  
 **def** clear(self, basic\_value=0):  
 *"""  
 Метод очистки буфера матрицы (заполняет содержимое переданным элементом).  
  
 Args:  
 basic\_value: целочисленное значение для заполнения матрицы.  
 """* self.a = [[basic\_value **for** j **in** range(self.n)] **for** i **in** range(self.n)]  
  
 **def** copy(self) -> Matrix:  
 *"""  
 Метод копирования объекта-матрицы.  
  
 Returns:  
 возвращает глубокую копию текущего объекта класса Matrix.  
 """* res = Matrix(self.n)  
 **for** i **in** range(self.n):  
 **for** j **in** range(self.n):  
 res.a[i][j] = self.a[i][j]  
 **return** res

**def** transponent(self) -> Matrix:  
 *"""  
 Метод транспонирования матрицы.  
  
 Returns:  
 Возвращает новый транспонированный объект типа Matrix.  
 """* T = self.copy()  
 **for** i **in** range(self.n):  
 **for** j **in** range(i):  
 T.a[i][j], T.a[j][i] = T.a[j][i], T.a[i][j]  
 **return** T  
  
 **def** generate(self,  
 a: int,  
 b: int,  
 h=**None**,  
 ah=**None**,  
 bh=**None**,  
 hstep=**None**) -> Tuple[List[float], List[List[float]]]:  
 *"""  
 Метод генерации симметричной матрицы с предопределенными собственными значениями.  
 Использует алгоритм: A = HhH где H = E - 2T(w)w - матрица Хаусхолдера.  
  
 Args:  
 a: целочисленная левая граница генерации вектора w  
 b: целочисленная правая граница генерации вектора w  
 h: вещественный вектор для заполнения его собственными значениями  
 ah: целочисленная левая граница генерации вектора h  
 bh: целочисленная правая граница генерации вектора h  
 hstep: шаг генерации вектора h  
  
 Returns:  
 Возвращает список собственных значений матрицы и список соответствующих им собственных векторов.  
 """  
 # Заполнение h* **if** bh **is None**:  
 bh = BP.bh  
 **if** ah **is None**:  
 ah = BP.ah  
 **if** hstep **is None**:  
 hstep = (bh - ah) / self.n  
 **if** abs(bh) == abs(ah):  
 hstep = (bh - ah - 1) / self.n  
 **if** h **is None**:  
 h = []  
 **if** h == []:  
 **for** k **in** range(self.n):  
 h.append(ah + k \* hstep)  
 *# Генерация* **for** i **in** range(self.n):  
 self.a[i][i] = h[i]  
 w = [random.randint(a, b + 1) **for** \_ **in** range(self.n)]  
 **for** i **in** range(len(w)):  
 **while** w[i] == 0:  
 w[i] = random.randint(a, b + 1)  
 w = normalize(w)  
 H = [[0 **for** i **in** range(self.n)] **for** j **in** range(self.n)]  
 **for** i **in** range(self.n):  
 **for** j **in** range(self.n):  
 **if** i == j:  
 H[i][j] = 1  
 H[i][j] -= 2\*w[i]\*w[j]  
 Hshd = Matrix(self.n)  
 Hshd.a = H  
 HshdT = Hshd.transponent()  
 self.a = (Hshd \* self \* HshdT).a  
 X = [[Hshd.a[j][i] **for** j **in** range(self.n)] **for** i **in** range(self.n)]  
 **return** h, X  
  
 **def** \_\_mul\_\_(self, x: object) -> object:  
 *"""  
 Переопредление оператора умножения в нескольких формах:  
 1. Matrix \* Matrix  
 2. Matrix \* int  
 3. Matrix \* List[float]  
  
 Args:  
 x: объект, на который умножается матрица (число, список, или другая матрица).  
  
 Returns:  
 Matrix или List[float] в зависимости от переданного значения.  
 """* **if** type(x) == float:  
 res = self.copy()  
 **for** i **in** range(self.n):  
 **for** j **in** range(self.n):  
 res.a[i][j] = self.a[i][j]\*x  
 **return** res  
 **if** type(x) == Matrix:  
 **if** x.n != self.n:  
 **raise** ValueError(**f"Несоразмерные матрицы: self.n = {**self.n**}, x.n = {**x.n**}"**)  
 res = Matrix(self.n)  
 **for** i **in** range(self.n):  
 **for** j **in** range(self.n):  
 **for** k **in** range(self.n):  
 res.a[i][j] += self.a[i][k] \* x.a[k][j]  
 **return** res  
 *# Проверяем условия когда перемножать нельзя* **if** type(x) != list **and** type(x) != tuple():  
 **raise** TypeError(**"Умножение возможно только в случае Matrix \* tuple/list"**)  
 **if** len(x) != self.n:  
 **raise** ValueError(**f"Размерность вектора должна быть = {**self.n**}"**)  
 *# Перемножаем* f = [0 **for** \_ **in** range(self.n)]  
 **for** i **in** range(self.n):  
 **for** k **in** range(self.n):  
 f[i] += self.a[i][k] \* x[k]  
 **return** f

1. **Модуль BaseParams:**

*"""Набор глобальных переменных для упрощенния запуска функций"""*cnt = 10 *# кол-во тестов*N = 6 *# размерность матрицы*a, b = -10, 10 *# границы генерации элементов вектора w*ah, bh = 1, 10 *# границы генерации собственных значений*matrix = **None** *# матрица для решения конкретной задачи пользователя*X = **None** *# список собственных векторов*h = **None** *# вектор собственных значений*by\_iter = **False** *# запуск конечного числа итераций*iter = 7 *# ограничение по итерациям*h\_eps = 0.0001 *# точность по h*x\_eps = 0.0001 *# точность по x*

1. **Модуль BaseFunctions:**

*"""Набор вспомогательных функций, используемых в решении"""***import** random  
**import** BaseParams **as** BP  
**from** typing **import** List, Tuple  
**import** MenuOptions  
  
**def** norm(w: List[float]) -> float:  
 *"""  
 Функция подсчета евклидовой нормы вектора.  
  
 Args:  
 w: вектор в вещественных координатах.  
  
 Returns:  
 вещественное число - евклидову норму переданного вектора.  
 """* **return** sum(map(**lambda** x: x\*\*2, w))\*\*0.5  
  
  
**def** normalize(w: List[float]) -> List[float]:  
 *"""  
 Функция нормирования переданного вещественного вектора по евклидовой номер.  
  
 Args:  
 w: вектор в вещественных координатах.  
  
 Returns:  
 новый вектор в вещественных координатах, нормированный по заданному вектору.  
 """* w\_norm = norm(w)  
 **return** list(map(**lambda** x: x / w\_norm, w))  
  
  
**def** get\_max\_abs\_index(lst: List[float]) -> int:  
 *"""  
 Функция поиска максимального по модулю элемента в списке.  
  
 Args:  
 lst: список вещественных чисел для поиска.  
  
 Returns:  
 целое число - индекс максимального по модулю элемента в переданном списке.  
 """* i\_abs\_max = 0  
 **for** i **in** range(1, len(lst)):  
 **if** abs(lst[i\_abs\_max]) < abs(lst[i]):  
 i\_abs\_max = i  
 **return** i\_abs\_max  
  
  
**def** get\_list\_without\_cnt\_of\_abs\_max(lst: List[float], cnt: int) -> List[float]:  
 *"""  
 Функция замещения элементов списка.  
 Замещает переданное количество максимальных по модулю элементов списка фиктивными 0.  
  
 Args:  
 lst: список вещественных чисел для обработки.  
 cnt: количество элементов для замещения.  
  
 Returns:  
 преобразованный список с фиктивными 0 элементами.  
 """* lst = lst[:]  
 **for** \_ **in** range(cnt):  
 lst[get\_max\_abs\_index(lst)]=0  
 **return** lst  
  
  
**def** corner(v1: List[float], v2: List[float]) -> float:  
 *"""  
 Функция анализа угла между векторами.  
 Находит косинус угла между векторами и абсолютную разность его отклонения от 1 (или угла от 0 угла).  
  
 Args:  
 v1: вещественный вектор для анализа.  
 v2: вещественний вектор для анализа.  
  
 Returns:  
 абсолютную разность отклонения косинуса угла от 1 (или угла от 0 угла).  
 """* res = sum([v1[i]\*v2[i] **for** i **in** range(BP.N)])  
 res /= norm(v1) \* norm(v2)  
 **return** abs(1-res)  
  
  
**def** exhaustion() -> Tuple[Tuple[int, float], List[float], float, int]:  
 *"""  
 Функция метода исчерпывания.  
 Находит 2 максимальное по модулю собственное значение матрицы, и соответсвующий ему собственный вектор.  
  
 Returns:  
 ( (индекс с.з., с.з), собственный вектор, оценку точности решения, количество выполненных итераций).  
 """* A1 = BP.matrix.copy()  
 tmp = get\_list\_without\_cnt\_of\_abs\_max(BP.h, 0)  
 ind = get\_max\_abs\_index(tmp)  
 hn = BP.h[ind]  
 xn = BP.X[ind]  
 **for** i **in** range(BP.N):  
 **for** j **in** range(BP.N):  
 A1.a[i][j] -= hn \* xn[i] \* xn[j]  
 tmp = get\_list\_without\_cnt\_of\_abs\_max(tmp, 1)  
 ind = get\_max\_abs\_index(tmp)  
 hn = BP.h[ind]  
 xn = BP.X[ind]  
 **for** i **in** range(BP.N):  
 **for** j **in** range(BP.N):  
 A1.a[i][j] -= hn \* xn[i] \* xn[j]  
 x\_cur = [0 **for** \_ **in** range(BP.N)]  
 **for** i **in** range(BP.N):  
 **while** x\_cur[i] == 0:  
 x\_cur[i] = random.randint(BP.a, BP.b + 1)  
 x\_cur = [A1.a[i][ind] **for** i **in** range(A1.n)]  
 i\_cur = 0  
 x\_prev = 0  
 b\_prev = 0  
 b\_cur = 0  
 **while** (BP.by\_iter **and** i\_cur < BP.iter) **or** \  
 (**not** BP.by\_iter **and** (i\_cur < 1 **or** abs(b\_prev - b\_cur) > BP.h\_eps **and** corner(x\_cur, x\_prev) > BP.x\_eps)):  
 v\_cur = normalize(x\_cur)  
 x\_prev = x\_cur  
 x\_cur = A1 \* v\_cur  
 b\_prev = b\_cur  
 b\_cur = sum([v\_cur[i] \* x\_cur[i] **for** i **in** range(BP.N)])  
 i\_cur += 1  
 tmp = get\_list\_without\_cnt\_of\_abs\_max(tmp, 1)  
 ind = get\_max\_abs\_index(tmp)  
 x\_cur = normalize(x\_cur)  
 r = 0  
 tmp = A1 \* x\_cur  
 **for** i **in** range(len(tmp)):  
 tmp[i] -= b\_cur \* x\_cur[i]  
 r = max(list(map(**lambda** x: abs(x), tmp)))  
 **return** (ind, b\_cur), x\_cur, r, i\_cur  
  
  
**def** rate(x: List[float], xt: List[float]) -> float:  
 *"""  
 Метод подсчета ошибки нахождения вектора-решения.  
  
 Args:  
 x: подсчитанный вектор.  
 xt: точный вектор.  
  
 Returns:  
 первую норму разности векторов.  
 """* size = len(x)  
 d = []  
 **for** i **in** range(size):  
 d.append(abs(x[i] - xt[i]))  
 **return** max(map(**lambda** x: abs(x), d))  
  
  
**def** average\_by\_test():  
 *"""  
 Метод тестирования по вычислению средних значений.  
 Запускается в случае ввода команды test.  
 Выводит на экран таблицу, содержащую все основные данные о работе тестирующей системы.  
 """* list\_h, list\_x, list\_r, list\_iter\_cnt = [], [], [], []  
 list\_h\_t, list\_x\_t = [], []  
 **for** i **in** range(BP.cnt):  
 **try**:  
 MenuOptions.gen()  
 (h\_ind, h), x, r, iter\_cnt = exhaustion()  
 list\_h.append(h) *# h подсчитанное* list\_h\_t.append(BP.h[h\_ind]) *# ht точное* list\_x.append(x[:]) *# x подсчитанный* list\_x\_t.append(BP.X[h\_ind][:]) *# xt точный* list\_r.append(r) *# r подсчитанный* list\_iter\_cnt.append(iter\_cnt) *# cnt\_iter подсчитанный* **except** Exception **as** e:  
 e.with\_traceback()  
  
 h\_aver = sum([abs(list\_h[i] - list\_h\_t[i]) **for** i **in** range(len(list\_h))]) / len(list\_h)  
 x\_aver = sum([rate(list\_x[i], list\_x\_t[i]) **for** i **in** range(len(list\_x))]) / len(list\_x)  
 r\_aver = sum(list\_r) / len(list\_r)  
 iter\_cnt\_aver = sum(list\_iter\_cnt) / len(list\_iter\_cnt)  
 string = **"| "** + str(BP.N).ljust(6) + **" |"** + \  
 **" "** + **f" [{**BP.ah**} ; {**BP.bh**}]"**.ljust(12) + **" |"** + \  
 **" "** + **f"{**BP.h\_eps**}"**.ljust(10) + **" |"** + \  
 **" "** + **f"{**h\_aver**}"**[:14].ljust(14) + **" |"** + \  
 **" "** + **f"{**x\_aver**}"**[:14].ljust(14) + **" |"** + \  
 **" "** + **f"{**r\_aver**}"**[:14].ljust(14) + **" |"** + \  
 **" "** + **f"{**iter\_cnt\_aver**}"**.ljust(8) + **" |"** *# символы: 11 12 11 18 17 17 14* print(**"+-------------+----------------+-------------------+------------------+------------------+------------------+--------------+"**,  
 **f"| Размерность | Диапазон h | h\_esp = x\_eps | Ср. оценка h | Ср. оценка x | Ср. оценка r | Cр. cnt\_iter |"**,  
 **"+-------------+----------------+-------------------+------------------+------------------+------------------+--------------+"**,  
 string,  
 **"+-------------+----------------+-------------------+------------------+------------------+------------------+--------------+"**,  
 sep=**'\n'**)

1. **Модуль MenuOptions:**

*"""  
Модуль настроек меню.  
Содержит одноименные командные функции для запуска из основной части программы.  
"""***from** Matrix **import** \*  
**from** BaseFunctions **import** \*  
**import** BaseParams **as** BP

**def** solve():  
 *"""  
 Функция, вызываемая в случае ввода команды solve.  
 Выводит на экран найденное собственное значение и соответствующий ему собственный вектор.  
 """* **if not** (BP.matrix **is None**) **and not** (BP.X **is None**) **and not** (BP.h **is None**):  
 (h\_ind, h), x, r, cnt\_iter = exhaustion()  
 print(**f"{**cnt\_iter**} итераций с точностью r = {**r**}"**,  
 **f"h{**h\_ind+1**} = {**h**}"**,  
 **f"x{**h\_ind+1**} = {**x**}"**,  
 sep=**'\n'**)