house_givens

10 марта 2023 г.

1 Семинар. Метод отражений Хаусхолдера

1.1 Импорт необходимых библиотек

[1]:
```
%matplotlib inline
%matplotlib notebook
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
import numpy as np
```

1.2 Отражения на плоскости

[2]:
```
from IPython.display import IFrame
IFrame("ink/HouseMirror2d.pdf", width=900, height=1600)
```

[2]:
```
<IPython.lib.display.IFrame at 0x1f7f3ccc810>
```

Определение. Ортогональная матрица $H \in \mathbb{R}^{2\times 2}$ называется *матрицей отражения*, если

$$H = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ \sin(\theta) & -\cos(\theta) \end{bmatrix}.$$

Если $y = Hx = H^T x$, то y получается отражением x относительно оси, определяемой как

$$S = \operatorname{span}\left\{ \begin{bmatrix} \cos\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right) \end{bmatrix} \right\}.$$

1.2.1 Пример

[3]:
```
theta = np.pi / 3; theta05 = 0.5 * theta
s0 = np.array([np.cos(theta05), np.sin(theta05)])
H = np.array([[np.cos(theta), np.sin(theta)],
              [np.sin(theta), -np.cos(theta)]])
x = np.array([5, 1])
y = np.matmul(H, x)
print(x,y)
```

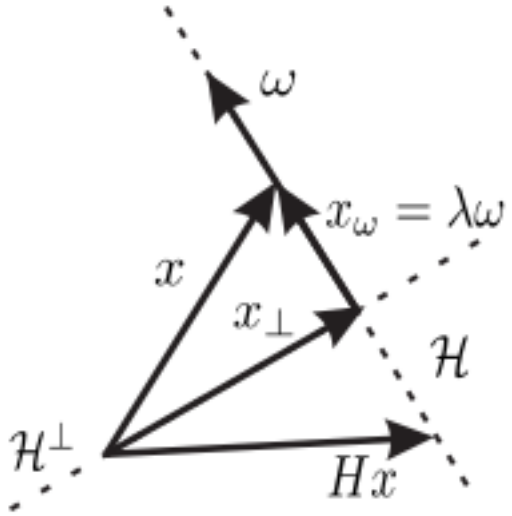
```
[5 1] [3.3660254  3.83012702]
```

[4]:
```
O = np.zeros(3)
X = np.array([s0[0], x[0], y[0]])
Y = np.array([s0[1], x[1], y[1]])

fig, ax = plt.subplots(figsize = (10, 10))
ax.quiver(O, O, X, Y,
          angles='xy', scale_units='xy', scale=1,
          color=['black', 'g', 'b'])

k = s0[1]/s0[0]

ax.plot([-1, 7], [-k, 7*k], 'r--', linewidth=1, label = '$S$')

ax.set_xlim(-1, 7)
ax.set_ylim(-1, 7)

ax.set_title('Преобразование отражения на плоскости')

from matplotlib.lines import Line2D

legend_elements = [
                    Line2D([0], [0], color='black', lw=2, label='$s0$'),
                    Line2D([0], [0], color='g', lw=2, label='$x$'),
                    Line2D([0], [0], color='b', lw=2, label='$y$'),
                    Line2D([0], [0], color='r', lw=2, label='$S$')
    ]
ax.legend(handles=legend_elements, loc='upper right')

plt.show()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

1.3 Многомерные отражения



let - $\omega \in \mathbb{R}^n$, $||\omega|| = 1$ — произвольный единичный вектор; - $\mathcal{H} = \text{span}\{\omega\}$ — одномерное линейное пространство; - $\mathcal{H}^\perp$ — ортогональное дополнение $\mathcal{H}$.

Тогда, $\forall x \in \mathbb{R}^n$: $x = x_\omega + x_\perp$, где - $x_\omega = \lambda\omega$ $(\lambda \in \mathbb{R})$ — ортогональная проекция вектора x на $\mathcal{H}$; - $x_\perp$ — его ортогональная составляющая.

Рассмотрим преобразование

$$Hx = H(x_\perp + x_\omega) = x_\perp - x_\omega = \{x_\perp = x - x_\omega\} = x - 2x_\omega = x - 2\lambda\omega$$

Т. к. $||\omega|| = 1 \Rightarrow \lambda = (x, \omega)$ и

$$Hx = x - 2(x,\omega)\omega = x - 2\omega(\omega, x) = x - 2\omega(\omega^T x) = x - 2(\omega\omega^T)x = \left(E - 2\omega\omega^T\right)x\,.$$

Определение. Оператор H называется *оператором отражений* или *оператором Хаусхолдера.* Матрицу $H = E - 2\omega\omega^T$, соответствующую данному оператору, называют *матрицей отражений* или *матрицей Хаусхолдера.*

let $v \in \mathbb{R}^n$ $(v \neq 0 \Rightarrow \omega = \frac{v}{||v||})$, тогда матрицу оператора Хаусхолдера можно записать в виде

$$\begin{aligned} Hx =& x - 2(x,\omega)\omega = x - 2\omega(x,\omega) = x - 2\omega(\omega, x) = \\ =& x - 2\frac{v(v,x)}{||v||^2} = x - \frac{2v(v^T x)}{v^T v} = \left(E - \frac{2vv^T}{v^T v}\right)x \Rightarrow \end{aligned}$$

$H = E - \dfrac{2vv^T}{v^T v}$, где вектор v называют *вектором Хаусхолдера.*

1.3.1 Свойства оператора Хаусхолдера

Преобразование Хаусхолдера - является линейным; - симметричный оператор

$$H^T = \left(E - 2\omega\omega^T\right)^T = E^T - 2(\omega\omega^T)^T = E - 2(\omega^T)^T\omega^T = E - 2\omega\omega^T = H\,;$$

- ортогональный оператор

$$H \cdot H^T = H \cdot H = \left(E - 2\omega\omega^T\right)\left(E - 2\omega\omega^T\right) = E - 2\omega\omega^T - 2\omega\omega^T + 4\omega\omega^T\omega\omega^T = \left\{\omega^T\omega = 1\right\} = E\,.$$

1.3.2 Собственные числа и векторы

- Из ортогональности оператора Хаусхолдера $\Rightarrow$ собственными числами оператора H являются ± 1;
- Собственные векторы:
  - Собственному значению -1 отвечает собственный вектор ω: $H\omega = E\omega - 2\omega\omega^T\omega = \left\{\omega^T\omega = 1\right\} = \omega - 2\omega = -1 \cdot \omega$;
  - Произвольный вектор $\nu \neq 0$, ортогональный ω, является собственным вектором H: $H\nu = E\nu - 2\omega\omega^T\nu = \left\{\omega^T\nu = 0\right\} = +1 \cdot \nu$, отвечающим собственному числу $+1$.

1.3.3 Альтернативное определение оператора Хаусхолдера

let оператор для некторого единичного вектора $\omega \in \mathbb{R}^n$ удовлетворяет условиям: - $H\omega = -1 \cdot \omega$; - $\forall \nu \neq 0$: $\nu \perp \omega \Rightarrow H\nu = +1 \cdot \nu$.

Произвольный вектор x можно представить в виде $x = x_\perp + x_\omega$, где - $(x_\perp, \omega) = 0$, - $x_\omega = \lambda\omega$

$\Rightarrow x_\omega = (x, \omega)\omega$, $x_\perp = x - (x, \omega)\omega$.

Тогда $Hx = H(x_\perp + x_\omega) = Hx_\perp + Hx_\omega$, а вектор $x_\perp$ ортогонален ω и, значит, является собственным вектором H, отвечающим собственному числу $+1$ $\Rightarrow$

$$\begin{aligned} Hx =& x_\perp + (x, \omega)H\omega = \{H\omega = -\omega\} = x_\perp - (x, \omega)\omega = \\ =& x_\perp - x_\omega = \{x_\perp = x - x_\omega\} = x - 2x_\omega \,. \end{aligned}$$

Значит задает зеркальное отражение вектора x относительно гиперплоскости, ортогональной вектору ω.

1.3.4 Пример

[5]:
```
x = np.array([7, 7, 1])
v = np.array([-1, -3, 5]); omega = v / np.linalg.norm(v)
y = x - 2 * np.dot(x, omega) * omega
print(x,y)
```

```
[7 7 1] [5.68571429 3.05714286 7.57142857]
```

[6]:
```
from mpl_toolkits.mplot3d import Axes3D

t = np.linspace(-10, 10, 11)
x_plane, y_plane = np.meshgrid(t, t)

z_plane = -(omega[0]*x_plane + omega[1]*y_plane)/omega[2]

fig = plt.figure(figsize = (10, 10))
ax = Axes3D(fig, auto_add_to_figure=False)

ax.plot_surface(x_plane, y_plane, z_plane, linewidth=0, rstride=1, cstride=1,␣
↪alpha=0.3, cmap='rainbow')

ax.quiver(0, 0, 0, v[0], v[1], v[2], color='black', label=r'$\omega$')
```

```
ax.quiver(0, 0, 0, x[0], x[1], x[2], color='g', label=r'$x$')
ax.quiver(0, 0, 0, y[0], y[1], y[2], color='b', label=r'$y$')

ax.set_title('Преобразование отражения в пространстве')

ax.legend(loc='upper left')
fig.add_axes(ax)

ax.elev = 15
ax.azim = 15

plt.show()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

1.3.5 Основная задача

Подобрать вектор ω в операторе Хаусхолдера таким образом, чтобы в результате преобразования полученный вектор имел направление заданного единичного вектора e ($||e|| = 1$), т. е. $Hx = \pm\alpha e$ ($\alpha > 0$).

$Hx = x - 2(x,\omega)\omega \Rightarrow Hx - x = -2(x,\omega)\omega = \pm\alpha e - x \Rightarrow \omega \in \text{span}\{x \pm \alpha e\} \Rightarrow \omega = \frac{x \pm \alpha e}{||x \pm \alpha e||} \Rightarrow$

$$
\begin{aligned}
Hx =& x - 2(x,\omega)\omega = x - 2\frac{(x, x \pm \alpha e)}{||x \pm \alpha e||^2}(x \pm \alpha e) = \\
=& x - \frac{2||x||^2 \pm 2\alpha(x,e)}{||x||^2 \pm 2\alpha(x,e) + \alpha^2||e||^2}(x \pm \alpha e) = \\
=& \left\{ \begin{array}{c} \text{При ортогональных преобразованиях длины векторов сохраняются} \Rightarrow \\ ||Hx|| = ||x|| = ||\alpha e|| = \alpha||e|| = \alpha \Rightarrow ||x|| = \alpha \end{array} \right\} = \\
=& x - \frac{2||x||^2 \pm 2\alpha(x,e)}{2||x||^2 \pm 2\alpha(x,e)}(x \pm \alpha e) = x - x \mp \alpha e = \mp\alpha e = \mp||x||e\,.
\end{aligned}
$$

На практике отражения Хаусхолдера применяются для обнуления всех компонент вектора начиная с некоторой

$$(x_1, \ldots, x_k, x_{k+1} \ldots, x_n) \overset{H}{\to} (y_1, \ldots, y_k, 0 \ldots, 0)\,.$$

Рассмотрим задачу обнуления всех компонент вектора, кроме первой, а потом обощим ее решение на более общий случай. let задан ненулевой вектор $x \in \mathbb{R}^n$, требуется чтобы вектор Hx был коллинеарен единичному вектору $e_1 = (1, 0, \ldots, 0) \in \mathbb{R}^n$.

let $v = x \pm \alpha_1 e_1 = x \pm ||x||e_1 \Rightarrow y = Hx = \left(E - \frac{2vv^T}{v^T v}\right)x = \mp\sqrt{x^T x} \cdot e_1$.

1.3.6 Выбор знака при вычислении вектора Хаусхолдера

Если x почти коллинеарен вектору e_1, то вектор

$$v = x - \text{sign}x_1\sqrt{x^T x}\, e_1$$

имеет малую норму. Поэтому возможно появление большой относительной ошибки при вычислении множителя $\frac{2}{v^T v}$.

1.3.7 Пример

Выполним преобразование Хаусхолдера вектора $x = [0.99, 10^{-6}, 10^{-3}]^T$, почти коллинеарного вектору e_1, двумя способами.

[7]:
```
x = np.array([0.99, 1.0e-6, 1.0e-3])
e1 = np.array([1.0, 0, 0])

v = x - (1.0 if x[0] >= 0 else -1.0) * np.linalg.norm(x) * e1
display(v, np.outer(v,v), 2.0 / np.dot(v,v), 2.0 / np.dot(v,v)*np.outer(v,v))
```

```
array([-5.05050881e-07,  1.00000000e-06,  1.00000000e-03])
```

```
array([[ 2.55076393e-13, -5.05050881e-13, -5.05050881e-10],
       [-5.05050881e-13,  1.00000000e-12,  1.00000000e-09],
       [-5.05050881e-10,  1.00000000e-09,  1.00000000e-06]])
```

```
1999997.4898503653
```

```
array([[ 5.10152145e-07, -1.01010049e-06, -1.01010049e-03],
       [-1.01010049e-06,  1.99999749e-06,  1.99999749e-03],
       [-1.01010049e-03,  1.99999749e-03,  1.99999749e+00]])
```

if выбрать

$$v = x + \text{sign} x_1 \sqrt{x^T x}\, e_1\,,$$

то подобных сложностей не возникает. Нетрудно заметить, что при таком выборе знака $||v||_\infty = |v_1|$.

[8]:
```
x = np.array([0.99, 1.0e-6, 1.0e-3])
e1 = np.array([1.0, 0, 0])

v = x + (1.0 if x[0] >= 0 else -1.0) * np.linalg.norm(x) * e1
display(v, 2.0 / np.dot(v,v))
```

```
array([1.98000051e+00, 1.00000000e-06, 1.00000000e-03])
```

```
0.5101516349208487
```

1.3.8 Хранение вектора Хаусхолдера

Полезно также придерживаться такой нормировки вектора v, что $v_1 = 1$. Данное условие можно выполнить, если первую компоненту взять равной единице, т. е. $v_1 = 1$, а все остальные компоненты получить как

$$v_i = \frac{v_i}{x_1 + \text{sign}\ x_1 \sqrt{x^T x}}.$$

Данное представление иногда упрощает алгоритмы, в которых нужно хранить вектор Хаусхолдера.

1.3.9 Алгоритм вычисления вектора Хаусхолдера

По $x \in \mathbb{R}^n$ приведенная функция вычисляет $v \in \mathbb{R}^n$, такой, что $v[0] = 1$ и все компоненты вектора $y = \left(E - \dfrac{2vv^T}{v^T v}\right) x$, кроме $y[0]$, равны нулю.

[9]:
```
def house(x):
    n=len(x); mu=np.linalg.norm(x); v=np.copy(x)
    if mu:
        beta=x[0]+(1.0 if x[0] >= 0 else -1.0)*mu
        v[1:]/=beta
    v[0]=1.0
    return v
```

1.3.10 Пример

Дан вектор $x = (3,\ 1,\ 5,\ 1)$. Найдем матрицу оператора Хаусхолдера, который отражает вектор x относительно гиперплоскости перпендикулярной вектору v таким образом, что $Hx = \alpha_1 e_1$, где $e_1 = (1,\ 0,\ 0,\ 0)$.

[10]:
```
x = np.array([3.0, 1.0, 5.0, 1.0])

n = len(x)

v = house(x)
omega = v / np.linalg.norm(v)
H = np.eye(n) - 2 * np.outer(omega, omega)

display(54*H, 9*v, H@x)
```

```
array([[-27.,  -9., -45.,  -9.],
       [ -9.,  53.,  -5.,  -1.],
       [-45.,  -5.,  29.,  -5.],
       [ -9.,  -1.,  -5.,  53.]])
```

```
array([9., 1., 5., 1.])
```

```
array([-6.00000000e+00,  4.44089210e-16,  1.77635684e-15,  4.44089210e-16])
```

С учетом нормировки вектор Хаусхолдера $v = (1,\ 1/9,\ 5/9,\ 1/9)^T$. Тогда матрица Хаусхолдера

$$H = \frac{1}{54}\begin{pmatrix} -27 & -9 & -45 & -9 \\ -9 & 53 & -5 & -1 \\ -45 & -5 & 29 & -5 \\ -9 & -1 & -5 & 53 \end{pmatrix}.$$

1.3.11 QR-разложение: преобразования Хаусхолдера

Определение. Представление матрицы $A \in \mathbb{R}^{m\times n}$ в виде произведения

$$A = QR,$$

где $Q \in \mathbb{R}^{m\times m}$ — ортогональная, $R \in \mathbb{R}^{m\times n}$ — верхняя треугольная. Здесь будем считать $m \geq n$. Если A имеет полный столбцовый ранг, то первые n столбцов матрицы Q образуют ортонормированный базис подпространства range$A$. Тем самым QR-разложение дает один из способов получения ортонормированного базиса для набора векторов.

Идея алгоритма let - $A \in \mathbb{R}^{5\times 5}$. - Матрицы Хаусхолдера H_1 и H_2 таковы, что

$$H_2H_1A = \begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \otimes & \times & \times \\ 0 & 0 & \otimes & \times & \times \\ 0 & 0 & \otimes & \times & \times \end{pmatrix}$$

Найдем матрицу Хаусхолдера $\widetilde{H}_3$ размера 3×3 такую, что

$$\widetilde{H}_3 \begin{pmatrix} \otimes \\ \otimes \\ \otimes \end{pmatrix} = \begin{pmatrix} \times \\ 0 \\ 0 \end{pmatrix}.$$

Если $H_3 = \mathrm{diag}(E_2, \widetilde{H}_3)$, то

$$H_3H_2H_1A = \begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times & \times \end{pmatrix}$$

Выполнив 4 таких шага, мы получаем верхнюю треугольную матрицу $H_4H_3H_2H_1A = R \Rightarrow Q^T = H_1H_2H_3H_4$.

Обобщая алгоритм на $m\times n$-матрицу $Q = H_1H_2 \dots H_{n-1}$ имеем $A = QR$, где Q — ортогональная матрица, а R — верхняя треугольная матрица.

1.3.12 Умножение на матрицы Хаусхолдера

Применяя преобразование Хаусхолдера к матрицам, очень важно учесть специальную структуру матрицы Хаусхолдера. Пусть A — матрица, $H = E - \dfrac{2vv^T}{v^Tv}$. Тогда

$$\begin{aligned} HA =& \left(E - \frac{2vv^T}{v^Tv}\right)A = A - \frac{2vv^TA}{v^Tv} = \\ =& A - \frac{2v(A^Tv)^T}{v^Tv} = A + v \cdot \left(\frac{-2}{v^Tv}A^Tv\right)^T = A + vw^T, \end{aligned}$$

где $\beta = -\dfrac{2}{v^Tv}$ и $w = \beta A^Tv$.

$$\begin{aligned} AH =& A\left(E - \frac{2vv^T}{v^Tv}\right) = A - \frac{2Avv^T}{v^Tv} = \\ =& A - \frac{2(Av)v^T}{v^Tv} = A + \left(\frac{-2}{v^Tv}Av\right) \cdot v^T = A + wv^T, \end{aligned}$$

где $w = \beta A v$.

Таким образом, хаусхолдерова модификация матрицы складывается из умножения матрицы на вектор и модификации внешним произведением векторов. Если бы мы не заметили этого и обращались бы с H как с матрицей общего вида, то объем работы возрос бы на порядок. Преобразование Хаусхолдера никогда не требует явного формирования матрицы Хаусхолдера. Следующие две функции формально подтверждают это.

1.3.13 Алгоритм умножения на матрицу Хаусхолдера слева

Для данной матрицы $A \in \mathbb{R}^{m \times n}$ и ненулевого вектора $v \in \mathbb{R}^m$, где $v[0] = 1$, следующий алгоритм строит на месте матрицы A матрицу $HA = A + vw^T$, где $\beta = -\dfrac{2}{v^T v}$ и $w = \beta A^T v$.

[11]:
```
def row_house(A,v):
    beta = -2.0 / np.dot(v,v)
    w = beta * np.matmul(A.T,v)
    A+=np.outer(v,w)
```

1.3.14 Алгоритм умножение на матрицу Хаусхолдера справа

Для данной матрицы $A \in \mathbb{R}^{m \times n}$ и ненулевого вектора $v \in \mathbb{R}^m$, где $v[0] = 1$, следующий алгоритм строит на месте матрицы A матрицу $AH = A + wv^T$, где $\beta = -\dfrac{2}{v^T v}$ и $w = \beta A v$.

[12]:
```
def col_house(A,v):
    beta = -2.0 / np.dot(v,v)
    w = beta * np.matmul(A,v)
    A+=np.outer(w,v)
```

1.3.15 QR-разложение: преобразование Хаусхолдера

По матрице $A \in \mathbb{R}^{m \times n}$ $(m \geq n)$, алгоритм находит хаусхолдеровы матрицы $H_1, \ldots H_{n-1}$: при $Q = H_1 \cdot \ldots \cdot H_{n-1}$ матрица $Q^T A = R$ верхняя треугольная. При этом содержательная часть (верхний треугольник) матрицы R записывается на место верхнего треугольника матрицы A, а содержательные компоненты векторов Хаусхолдера записываются в нижней части.

[13]:
```
def house_QR(A):
    m,n=A.shape
    for j in range(n):
        v=house(A[j:,j])
        row_house(A[j:,j:],v)
        if j<m-1:
            A[j+1:,j]=v[1:]
```

[14]:
```
A=np.array([[1, 2, 3, 4], [7, 8, 9, 10], [12, 13, 14, 15], [16, 17, 18, 19],␣
 ↪[19, 20, 21, 22]],dtype='float64')

display(A)
```

```
house_QR(A)

display(A)
```

```
array([[ 1.,  2.,  3.,  4.],
       [ 7.,  8.,  9., 10.],
       [12., 13., 14., 15.],
       [16., 17., 18., 19.],
       [19., 20., 21., 22.]])
```

```
array([[-2.84780617e+01, -3.04093729e+01, -3.23406842e+01,
        -3.42719954e+01],
       [ 2.37464731e-01, -1.12695918e+00, -2.25391836e+00,
        -3.38087754e+00],
       [ 4.07082396e-01, -1.35081707e-01,  2.09476461e-15,
         4.23660259e-15],
       [ 5.42776528e-01, -4.13066445e-01,  5.24751609e-01,
        -4.10004660e-15],
       [ 6.44547127e-01, -6.21554998e-01, -6.12210211e-01,
         3.40110954e-01]])
```

1.3.16 Факторизованное представление матрицы Q

В алгоритмах разложения, основанных на преобразованиях отражения, вычисляется произведение матриц Хаусхолдера

$$Q = H_1 \cdot \ldots \cdot H_r, \quad H_j = E - 2\frac{v^{(j)}{v^{(j)}}^T}{{v^{(j)}}^T v^{(j)}},$$

где $r \leq n$ и каждое $v^{(j)} = (\underbrace{0, \ldots, 0}_{j-1}, 1, v_{j+1}^{(j)}, \ldots, v_n^{(j)})^T$.

Явное вычисление матрицы Q обычно не требуется, даже если Q используется в последующих вычислениях. Например, если $C \in \mathbb{R}^{n \times \ell}$ требуется вычислить CQ

```
for j in range(r):
    v[j]=1; v[j+1:]=A[j+1:,j]
    col_house(C[j:,:],v[j:])
```

В некоторых приложениях требуется явное формирование матрицы Q. Для вычисления Q возможны два алгоритма: - прямое накопление; - обратное накопление.

Алгоритм прямого накопления

$$B = E, \quad Q = (\ldots(B \cdot H_1) \cdot \ldots) \cdot H_{n-1}$$

[15]:
```
def house_dir_accum(A,B):
    m,n=A.shape
    v=np.zeros(m)
    for j in range(n):
```

```
        v[:j]=0.0; v[j]=1.0; v[j+1:]=A[j+1:,j]
        col_house(B,v)
```

Алгоритм обратного накопления

$$B = E\,, \quad Q = H_1 \cdot (\ldots \cdot (H_{n-1} \cdot B)\ldots)$$

[16]:
```
def house_rev_accum(A,B):
    m,n=A.shape
    v=np.zeros(m)
    for j in range(n-1,-1,-1):
        v[j]=1.0; v[j+1:]=A[j+1:,j]
        row_house(B[j:,j:],v[j:])
```

Пример Вычислить QR разложение матрицы $A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 7 & 8 & 9 & 10 \\ 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 \\ 19 & 20 & 21 & 22 \end{bmatrix}$.

[17]:
```
A=np.array([[1, 2, 3, 4], [7, 8, 9, 10], [12, 13, 14, 15], [16, 17, 18, 19],␣
 ↪[19, 20, 21, 22]],dtype='float64')
house_QR(A)

m,n=A.shape
Q1=np.eye(m); Q2=np.eye(m)

house_dir_accum(A,Q1)
house_rev_accum(A,Q2)

Q1-Q2
```

[17]:
```
array([[ 0.00000000e+00,  1.11022302e-16, -6.93889390e-18,
         0.00000000e+00, -1.11022302e-16],
       [ 0.00000000e+00,  3.33066907e-16, -8.32667268e-17,
         2.22044605e-16, -8.32667268e-17],
       [ 0.00000000e+00,  2.77555756e-17, -1.11022302e-16,
         0.00000000e+00,  0.00000000e+00],
       [ 0.00000000e+00,  1.11022302e-16,  0.00000000e+00,
         8.32667268e-17,  1.11022302e-16],
       [ 0.00000000e+00,  1.11022302e-16, -1.11022302e-16,
        -8.32667268e-17,  8.32667268e-17]])
```

[18]:
```
A=np.array([[1, 2, 3, 4], [7, 8, 9, 10], [12, 13, 14, 15], [16, 17, 18, 19],␣
 ↪[19, 20, 21, 22]],dtype='float64')
house_QR(A)
```

```
m,n=A.shape
Q=np.eye(m)
house_rev_accum(A,Q)

R=np.zeros((m,n))

for j in range(n):
    R[:j+1,j]=A[:j+1,j]

display(Q@R)
```

```
array([[ 1.,  2.,  3.,  4.],
       [ 7.,  8.,  9., 10.],
       [12., 13., 14., 15.],
       [16., 17., 18., 19.],
       [19., 20., 21., 22.]])
```

Вопрос. Почему применение обратного накопления эффективнее?

2 Семинар. Метод вращений Гивенса

2.1 Вращения на плоскости

Определение. Ортогональная $G \in \mathbb{R}^{2\times 2}$ называется *матрицей вращения*, если

$$G = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}.$$

Если $y = Gx$, то y получается поворотом x на угол θ против часовой стрелки.

Отражения Хаусхолдера полезны при крупномасштабных обнулениях. Однако, когда необходимо избирательное зануление элементов предпочтительнее использовать *вращения Гивенса*.

2.2 Многомерные вращения

Определение. Ортогональная матрица $G \in \mathbb{R}^{n\times n}$ вида

$$G = \begin{pmatrix} 1 & \dots & 0 & \dots & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \dots & c & \dots & -s & \dots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \dots & s & \dots & c & \dots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & \dots & 0 & \dots & 1 \end{pmatrix} \begin{matrix} \\ \\ i \\ \\ k \\ \\ \\ \end{matrix},$$

где $c = \cos(\theta)$, а $s = \sin(\theta)$ называется матрицей Гивенса.

Если $y = Gx$, то $y_j = \begin{cases} cx_i - sx_k, & j = i, \\ sx_i + cx_k, & j = k, \\ x_j, & j \neq k, i. \end{cases}$.

Тогда $y_k = sx_i + cx_k = 0$, выбрав $\cos(\theta) = c = \dfrac{x_i}{\sqrt{x_i^2 + x_k^2}}$, $\sin(\theta) = s = \dfrac{-x_k}{\sqrt{x_i^2 + x_k^2}}$.

Данное преобразование равносильно повороту вектора x на угол θ против часовой стрелки в координатной плоскости (i, k).

2.2.1 Алгоритм. Вычисление элементов матрицы поворота

По скалярам a и b эта функция определяет $c = \cos\theta$ и $s = \sin\theta$ так, что

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix} \Rightarrow \begin{cases} s = \dfrac{|b|}{\sqrt{a^2+b^2}}, c = \dfrac{-a}{b\sqrt{1+(a/b)^2}}, |b| > |a| \\ c = \dfrac{|a|}{\sqrt{a^2+b^2}}, s = \dfrac{-b}{a\sqrt{1+(b/a)^2}}, |b| \le |a| \end{cases}.$$

[19]:
```
def givens(a,b):
    if b==0:
        c=1; s=0
    else:
        if np.abs(b)>np.abs(a):
            tau=-a/b; s=1/np.sqrt(1+tau**2); c=s*tau
        else:
            tau=-b/a; c=1/np.sqrt(1+tau**2); s=c*tau
    return c,s
```

2.2.2 Алгоритм. Умножение на матрицу поворота слева

Пусть $A \in \mathbb{R}^{2\times n}$, $= \cos\theta$, $s = \sin\theta$. Следующий алгоритм записывает на место A матрицу $\begin{bmatrix} c & -s \\ s & c \end{bmatrix} \cdot A$.

[20]:
```
def row_rot(A,c,s):
    a0=np.array(A[0,:]); a1=np.array(A[1,:])
    A[0,:]=c*a0[:]-s*a1[:]
    A[1,:]=s*a0[:]+c*a1[:]
```

2.2.3 Алгоритм. Умножение на матрицу поворота справа

Пусть $A \in \mathbb{R}^{m\times 2}$, $= \cos\theta$, $s = \sin\theta$. Следующий алгоритм записывает на место A матрицу $A \cdot \begin{bmatrix} c & -s \\ s & c \end{bmatrix}$.

[21]:
```
def col_rot(A,c,s):
    a0=np.array(A[:,0]); a1=np.array(A[:,1])
    A[:,0]=c*a0[:]+s*a1[:]
    A[:,1]=-s*a0[:]+c*a1[:]
```

2.2.4 QR-разложение: преобразование Гивенса

Преобразования Гивенса, так же как и Хаусхолдера можно использовать для QR-разложения матрицы. Т. е. представления матрицы A в виде $A = QR$, где Q — ортогональная матрица, а R — верхняя треугольная матрица.

Иллюстрация на примере 3×3-матрицы

$$\begin{pmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{pmatrix} \longrightarrow \begin{pmatrix} \times & \times & \times \\ 0 & \times & \times \\ \times & \times & \times \end{pmatrix} \longrightarrow \begin{pmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \end{pmatrix} \longrightarrow \begin{pmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \end{pmatrix}$$

В общем случае, очевидно, если обозначить j-е вращение в этом приведении за G_j, то матрица $Q^T A$ будет верхней треугольной, где $Q = G_1^T \cdot \ldots \cdot G_r^T$ и r — общее количество вращений.

2.2.5 Алгоритм QR-разложение: преобразования Гивенса

По матрице $A \in \mathbb{R}^{m \times n}$ $(m \geq n)$, следующий алгоритм записывает на место A верхнюю треугольную матрицу $R = Q^T A$, где Q ортогональна.

[22]:
```
def givens_QR(A,store_rot=False):
    m,n=A.shape
    for j in range(n):
        for i in range(m-1,j,-1):
            c,s=givens(A[i-1,j],A[i,j])
            row_rot(A[i-1:i+1,j:],c,s)
            if store_rot:
                A[i,j]=np.arctan2(s,c)
```

Алгоритм обратного накопления

$$Q = G_1^T \cdot (\ldots \cdot (G_r^T \cdot E) \ldots)$$

[23]:
```
def givens_rev_accum(A):
    m,n=A.shape
    Q=np.eye(m)
    for j in range(n-1,-1,-1):
        for i in range(j,m-1,1):
            theta=A[i+1,j]; c=np.cos(theta); s=np.sin(theta)
            row_rot(Q[i:i+2,j:],c,-s)
    return Q
```

Пример Вычислить QR разложение матрицы $A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 7 & 8 & 9 & 10 \\ 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 \\ 19 & 20 & 21 & 22 \end{bmatrix}$.

[24]:
```
A=np.array([[1, 2, 3, 4], [7, 8, 9, 10], [12, 13, 14, 15], [16, 17, 18, 19],␣
 ↪[19, 20, 21, 22]],dtype='float64')
givens_QR(A,True)

Q=givens_rev_accum(A)

m,n=A.shape
R=np.zeros((m,n))

for j in range(n):
    R[:j+1,j]=A[:j+1,j]

Q,R,Q@R
```

[24]:
```
(array([[ 0.03511475,  0.82716615, -0.46211858, -0.10449298,  0.3001512 ],
        [ 0.24580325,  0.46610156,  0.3821894 , -0.08483883, -0.75436313],
        [ 0.42137699,  0.1652144 ,  0.40608361,  0.70453431,  0.36588679],
        [ 0.56183599, -0.07549532,  0.29642547, -0.67760017,  0.36280948],
        [ 0.66718024, -0.25602762, -0.6225799 ,  0.16239768, -0.27448434]]),
 array([[2.84780617e+01, 3.04093729e+01, 3.23406842e+01, 3.42719954e+01],
        [0.00000000e+00, 1.12695918e+00, 2.25391836e+00, 3.38087754e+00],
        [0.00000000e+00, 0.00000000e+00, 2.42539902e-15, 1.06722639e-15],
        [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 1.23925758e-15],
        [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00]]),
 array([[ 1.,  2.,  3.,  4.],
        [ 7.,  8.,  9., 10.],
        [12., 13., 14., 15.],
        [16., 17., 18., 19.],
        [19., 20., 21., 22.]]))
```

Домашнее задание. Разработать функцию `givens_dir_accum(A)`, реализующую прямое накопление. Какой из алгоримов накопления эффективнее и почему?

3 Семинар. Применение QR-разложений для решения СЛАУ. Решение треугольных систем

Очевидным применением QR-разложений является нахождение решений СЛАУ.

Рассмотрим СЛАУ

$$Ax = b; \quad A \in \mathbb{R}^{n\times n}; \; x, b \in \mathbb{R}^n.$$

Пусть $A = QR$, где $Q \in \mathbb{R}^{n\times n}$ — ортогональная, а $R \in \mathbb{R}^{n\times n}$ — верхняя треугольная матрица. Тогда следующая цепочка рассуждений

$$Ax = b \Leftrightarrow (QR)x = b \Leftrightarrow (Q^TQ)Rx = Q^Tb \Leftrightarrow Rx = Q^Tb$$

приводит к СЛАУ с верхней треугольной матрицей.

3.1 Решение треугольных систем

Традиционные методы разложений для линейных систем включают в себя приведение исходной квадратной системы к треугольной системе, решение которой совпадает с исходной. Данный раздел посвящен решению треугольных систем.

3.1.1 Прямая подстановка

Рассмотрим следующую нижнюю треугольную 2×2-систему:

$$\begin{bmatrix} l_{11} & 0 \\ l_{21} & l_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

Если $l_{11}l_{22} \neq 0$, то неизвестные могут быть определены последовательно:

$$\begin{aligned} x_1 &= \tfrac{b_1}{l_{11}}, \\ x_2 &= \tfrac{b_2 - l_{21}x_1}{l_{22}}. \end{aligned}$$

Это 2×2-версия алгоритма, известного как прямая подстановка. Общую процедуру получаем, разрешая i-е уравнение системы $Lx = b$ относительно :

$$x_i = \frac{\left(b_i - \sum\limits_{j=1}^{i-1} l_{ij}x_j\right)}{l_{ii}}.$$

Если выполнить для $i = \overline{1,n}$, то будут вычислены все компоненты x.

Вычисление $\sum\limits_{j=1}^{i-1} l_{ij}x_j$ можно интерпретировать, как вычисление скалярного произведения $L[i,:i] \cdot x[:i]$.

Поскольку b_i содержится только в формуле для x_i, на месте b_i можно записать x_i.

3.1.2 Алгоритм (Прямая подстановка: строчная версия)

Предположим, что $L \in \mathbb{R}^{n\times n}$ – нижняя треугольная матрица и $b \in \mathbb{R}^n$. Алгоритм заменяет b на решение системы $Lx = b$. Матрица L должна быть невырожденна.

[25]:
```
def dir_subs(L,b):
    n=len(b)
    b[0]/=L[0,0]
    for i in range(1,n):
        b[i]=(b[i]-np.dot(L[i,:i],b[:i]))/L[i,i]
```

На выполнение алгоритма затрачивается $O(n^2)$ флопов.

Пример Найти решение СЛАУ:

$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 1 & 7 & 0 & 0 \\ 1 & 0 & 3 & 0 \\ -1 & 1 & -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 4 \\ 9 \\ 2 \\ -3 \end{bmatrix}.$$

Аналитическое решение: $x = [2, 1, 0, -1]^T$

[26]:
```
L=np.array([[2, 0, 0, 0], [1, 7, 0, 0], [1, 0, 3, 0], [-1, 1, -1,␣
↪2]],dtype='float64')
b=np.array([4, 9, 2, -3],dtype='float64')

dir_subs(L,b)

display(b)
```

```
array([ 2.,  1.,  0., -1.])
```

3.1.3 Обратная подстановка

Аналогичный алгоритм для верхней треугольной системы $Ux = b$ называется *обратной подстановкой* $x_i = \frac{b_i - \sum\limits_{j=i+1}^{n} u_{ij}x_j}{u_{ii}}$ и снова x можно записать на месте b.

3.1.4 Алгоритм (Обратная подстановка: строчная версия)

Если матрица $U \in \mathbb{R}^{n\times n}$ верхняя треугольная и $b \in \mathbb{R}^n$, то следующий алгоритм заменяет b на решение системы $Ux = b$, U должна быть невырожденной.

[27]:
```
def rev_subs(U,b):
    n=len(b)
    b[n-1]/=U[n-1,n-1]
    for i in range(n-2,-1,-1):
        b[i]=(b[i]-np.dot(U[i,i+1:],b[i+1:]))/U[i,i]
```

Алгоритм требует $O(n^2)$ флопов и осуществляет доступ к U по строкам.

Пример Найти решение СЛАУ:

$$\begin{bmatrix} 1 & 2 & -1 & -2 \\ 0 & 3 & 0 & 1 \\ 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} -2 \\ 7 \\ 7 \\ 4 \end{bmatrix}.$$

Аналитическое решение: $x = [-1, 2, 3, 1]^T$

[28]:
```
U=np.array([[1, 2, -1, -2], [0, 3, 0, 1], [0, 0, 2, 1], [0, 0, 0,␣
↪3]],dtype='float64')
b=np.array([-2, 7, 7, 3],dtype='float64')

rev_subs(U,b)

display(b)
```

```
array([-1.,  2.,  3.,  1.])
```

3.1.5 Столбцовые версии

Столбцовые версии приведенных выше процедур могут быть получены изменением порядка выполнения циклов. Чтобы понять, что это означает с алгебраической точки зрения, рассмотрим прямую подстановку.

После нахождения x_1 ее можно удалить из всех уравнений $\overline{2,n}$, и продолжить процесс с редуцированной системой

$$L[2:n,2:n]x[2:n] = b[2:n] - x[1]L[2:n,1]\,.$$

На следующем шаге вычисляется $[2]$ и ее тоже можно исключить из всех уравнений $\overline{3,n}$, и т. д.

3.1.6 Пример

$$\begin{bmatrix} 2 & 0 & 0 \\ 1 & 5 & 0 \\ 7 & 9 & 8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ 2 \\ 5 \end{bmatrix}$$

Отсюда $x_1 = 3$ и потом имеем дело с 2×2-системой

$$\begin{bmatrix} 5 & 0 \\ 9 & 8 \end{bmatrix} \begin{bmatrix} x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 5 \end{bmatrix} - 3 \begin{bmatrix} 1 \\ 7 \end{bmatrix} = \begin{bmatrix} -1 \\ -16 \end{bmatrix}\,.$$

3.1.7 Алгоритм (Прямая подстановка: столбцовая версия)

Предположим, что $L \in \mathbb{R}^{n\times n}$ нижняя треугольная и $b \in \mathbb{R}^n$. Этот алгоритм заменяет b на решение системы $Lx = b$, матрица L должна быть невырожденной

[29]:
```
def dir_subs_cols(L,b):
    n=len(b)
    for j in range(n-1):
        b[j]/=L[j,j]
        b[j+1:]=b[j+1:]-b[j]*L[j+1:,j]
    b[n-1]/=L[n-1,n-1]
```

Пример Найти решение СЛАУ:

$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 1 & 7 & 0 & 0 \\ 1 & 0 & 3 & 0 \\ -1 & 1 & -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 4 \\ 9 \\ 2 \\ -3 \end{bmatrix}\,.$$

Аналитическое решение: $x = [2,1,0,-1]^T$

[30]:
```
L=np.array([[2, 0, 0, 0], [1, 7, 0, 0], [1, 0, 3, 0], [-1, 1, -1,␣
 ↪2]],dtype='float64')
b=np.array([4, 9, 2, -3],dtype='float64')

dir_subs_cols(L,b)
```

```
display(b)
```

```
array([ 2.,  1.,  0., -1.])
```

3.1.8 Алгоритм (Обратная подстановка: столбцовая версия)

Предположим, что $U \in \mathbb{R}^{n\times n}$ – верхняя треугольная матрица и $b \in \mathbb{R}^n$. Алгоритм заменяет b на решение $Ux = b$, матрица U должна быть невырождена

[31]:
```
def rev_subs_cols(U,b):
    n=len(b)
    for j in range(n-1,0,-1):
        b[j]/=U[j,j]
        b[:j]=b[:j]-b[j]*U[:j,j]
    b[0]/=U[0,0]
```

Пример Найти решение СЛАУ:

$$\begin{bmatrix} 1 & 2 & -1 & -2 \\ 0 & 3 & 0 & 1 \\ 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} -2 \\ 7 \\ 7 \\ 4 \end{bmatrix}.$$

Аналитическое решение: $x = [-1, 2, 3, 1]^T$

[32]:
```
U=np.array([[1, 2, -1, -2], [0, 3, 0, 1], [0, 0, 2, 1], [0, 0, 0,␣
 ↪3]],dtype='float64')
b=np.array([-2, 7, 7, 3],dtype='float64')

rev_subs_cols(U,b)

display(b)
```

```
array([-1.,  2.,  3.,  1.])
```

Чаще всего точность решения треугольной системы на удивление хорошая.

3.1.9 Применение QR-разложений для решения СЛАУ

Пример Найти решение СЛАУ:

$$\begin{bmatrix} 2 & 1 & 2 & 1 \\ 5 & 0 & -3 & -1 \\ 7 & 1 & 7 & 2 \\ 0 & 2 & 1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 9 \\ 0 \end{bmatrix}.$$

Аналитическое решение: $x = [1, -2, 0, 2]^T$

Напрашивается непосредственное применение QR-разложения. Выберем версию на основе преобразования Хаусхолдера.

[33]:
```
A=np.array([[2, 1, 2, 1], [5, 0, -3, -1], [7, 1, 7, 2], [0, 2, 1,␣
↪2]],dtype='float64')
b=np.array([2, 3, 9, 0],dtype='float64')

house_QR(A)

m,n=A.shape
Q=np.eye(m)

house_rev_accum(A,Q)

b1=Q.T@b
rev_subs(A,b1)

display(b1)
```

```
array([ 1.00000000e+00, -2.00000000e+00, -2.15643791e-15,  2.00000000e+00])
```

Однако, это решение неэффективно, поскольку требует явного формирования матрицы Q, без чего можно обойтись.

Алгоритм умножения вектора на матрицу Хаусхолдера слева Для данного вектора $x \in \mathbb{R}^n$ и ненулевого $v \in \mathbb{R}^n$, где $v[0] = 1$, следующий алгоритм строит на месте вектора x вектор $Px = \left(E - 2\frac{vv^T}{v^Tv}\right)x = x - \left(2\frac{v^Tx}{v^Tv}\right) \cdot v$.

[34]:
```
def row_house_vec(x,v):
    beta = -2.0 * np.dot(v,x) / np.dot(v,v)
    x += beta * v
```

Алгоритм решения СЛАУ, основанный на преобразовании Хаусхолдера Для невырожденной матрицы $A \in \mathbb{R}^{n\times n}$ и вектора $b \in \mathbb{R}^n$, следующий алгоритм, с помощью QR-разложения на основе преобразования Хаусхолдера, приводит СЛАУ $Ax = b$ к верхней треугольной, находит ее решение и помещает его в вектор b.

[35]:
```
def solve_house(A,b):
    house_QR(A)

    _,n=A.shape
    v=np.zeros(n)
    for j in range(n):
        v[j]=1.0; v[j+1:]=A[j+1:,j]
        row_house_vec(b[j:],v[j:])

    rev_subs(A,b)
```

Пример Найти решение СЛАУ:

$$\begin{bmatrix} 2 & 1 & 2 & 1 \\ 5 & 0 & -3 & -1 \\ 7 & 1 & 7 & 2 \\ 0 & 2 & 1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 9 \\ 0 \end{bmatrix}.$$

Аналитическое решение: $x = [1, -2, 0, 2]^T$

Численное решение с помощью непосредственного применения QR-разложения на основе преобразования Хаусхолдера:

```
array([ 1.00000000e+00, -2.00000000e+00, -2.15643791e-15,  2.00000000e+00])
```

[36]:
```
A=np.array([[2, 1, 2, 1], [5, 0, -3, -1], [7, 1, 7, 2], [0, 2, 1,␣
 ↪2]],dtype='float64')
b=np.array([2, 3, 9, 0],dtype='float64')

solve_house(A,b)

display(b)
```

```
array([ 1.00000000e+00, -2.00000000e+00, -1.78463827e-15,  2.00000000e+00])
```

Алгоритм умножения вектора на матрицу Гивенса слева Пусть $x \in \mathbb{R}^2$, $= \cos\theta$, $s = \sin\theta$. Следующий алгоритм записывает на место x вектор $y = \begin{bmatrix} cx_1 - sx_2 \\ sx_1 + cx_2 \end{bmatrix}$.

[37]:
```
def row_rot_vec(x,c,s):
    x0=x[0]; x1=x[1]
    x[0]=c*x0-s*x1
    x[1]=s*x0+c*x1
```

Алгоритм решения СЛАУ, основанный на преобразовании Гивенса Для невырожденной матрицы $A \in \mathbb{R}^{n\times n}$ и вектора $b \in \mathbb{R}^n$, следующий алгоритм, с помощью QR-разложения на основе преобразования Гивенса, приводит СЛАУ $Ax = b$ к верхней треугольной, находит ее решение и помещает его в вектор b.

[38]:
```
def solve_givens(A,b):
    givens_QR(A,True)

    _,n=A.shape
    for j in range(n):
        for i in range(n-1,j,-1):
            theta=A[i,j]; c=np.cos(theta); s=np.sin(theta)
            row_rot_vec(b[i-1:i+1],c,s)

    rev_subs(A,b)
```

Пример Найти решение СЛАУ:

$$\begin{bmatrix} 2 & 1 & 2 & 1 \\ 5 & 0 & -3 & -1 \\ 7 & 1 & 7 & 2 \\ 0 & 2 & 1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 9 \\ 0 \end{bmatrix}.$$

Аналитическое решение: $x = [1, -2, 0, 2]^T$

Численное решение с помощью непосредственного применения QR-разложения на основе преобразования Хаусхолдера:

```
array([ 1.00000000e+00, -2.00000000e+00, -2.15643791e-15,  2.00000000e+00])
```

Численное решение с помощью эффективного применения QR-разложения на основе преобразования Хаусхолдера:

```
array([ 1.00000000e+00, -2.00000000e+00, -1.78463827e-15,  2.00000000e+00])
```

[39]:
```
A=np.array([[2, 1, 2, 1], [5, 0, -3, -1], [7, 1, 7, 2], [0, 2, 1,␣
 ↪2]],dtype='float64')
b=np.array([2, 3, 9, 0],dtype='float64')

solve_givens(A,b)

display(b)
```

```
array([ 1.00000000e+00, -2.00000000e+00,  7.43599278e-17,  2.00000000e+00])
```