NI-MPI úkol

January 4, 2021

Vypracoval Jan Bittner

1 Kód

[215]:
```
import numpy as np
```

[272]:
```
class Solver:
    def __init__(self, matrix, vector, initialVector, precision, gamma):
        self.initialVector = initialVector
        self.precision = precision
        self.matrix = matrix
        self.bVector = vector
        self.gamma = gamma

        # lower triangular part
        self.l = np.tril(matrix, -1)

        # upper triangular part
        self.u = np.triu(matrix, 1)

        # diagonal component
        self.d = np.diag(np.diag(matrix))

        # init Q - must be set by subclases
        self.q = None
        self.qinv = None

    def solve(self):
        """Starts to compute iterations and then returns count of iterations␣
↪and result."""
        iterationCount = 0
        x = None

        if self.canConverge():
            x = self.initialVector

            while self.isNotPreciseEnough(x):
```

```
                iterationCount = iterationCount + 1
                x = self.doIteration(x)

        return iterationCount, x

    def canConverge(self):
        """Can converge if the value of spectral radius is less than 1."""
        e = np.identity(self.matrix.shape[0], dtype = np.float64)
        return self.getSpectralRadius(e - self.qinv @ self.matrix) < 1

    def isNotPreciseEnough(self, iteration):
        """Chech whether precision is not already sufficient."""
        return (np.linalg.norm(self.matrix @ iteration - self.bVector) / np.
↪linalg.norm(self.bVector)) > self.precision

    def doIteration(self, lastIteration):
        """Does next iteration."""
        return self.qinv @ (self.q - self.matrix) @ lastIteration + self.qinv @␣
↪self.bVector

    def getSpectralRadius(self, matrix):
        """Returns max absolute eigenvalue of matrix, aka spectral radius."""
        return max(abs(np.linalg.eigvals(matrix)))


class JacobiSolver(Solver):
    def __init__(self, matrix, vector, initialVector, precision, gamma):
        super().__init__(matrix, vector, initialVector, precision, gamma)
        self.q = self.d
        self.qinv = np.linalg.inv(self.q)


class GaussSeidelSolver(Solver):
    def __init__(self, matrix, vector, initialVector, precision, gamma, omega =␣
↪1):
        super().__init__(matrix, vector, initialVector, precision, gamma)
        self.omega = omega
        self.q = (1 / omega) * self.d + self.l
        self.qinv = np.linalg.inv(self.q)
```

[328]:
```
### ----- config

# parameters
gamma = 3
omega = 1
precision = 10**-6
```

```
# matrix
matrix = np.zeros((20, 20), dtype = np.float64)
np.fill_diagonal(matrix, gamma)
np.fill_diagonal(matrix[:, 1:], -1) # upper part
np.fill_diagonal(matrix[1:, :], -1) # lower part

# vector b
bVector = np.full((20, 1), gamma - 2, dtype = np.float64)
bVector[0] = bVector[0] + 1
bVector[-1] = bVector[-1] + 1

# initial vector
initialVector = np.zeros(bVector.shape, dtype = np.float64)

### ----- solver

# use one of these:
#solver = JacobiSolver(matrix, bVector, initialVector, precision, gamma)
solver = GaussSeidelSolver(matrix, bVector, initialVector, precision, gamma,␣
 ↪omega)

solver.solve()
```

[328]:
```
(33,
 array([[0.9999987 ],
        [0.99999898],
        [1.0000001 ],
        [1.00000059],
        [1.00000045],
        [1.00000026],
        [1.00000023],
        [1.00000027],
        [1.00000026],
        [1.00000018],
        [1.0000001 ],
        [1.00000004],
        [1.00000001],
        [1.        ],
        [0.99999999],
        [0.99999998],
        [0.99999998],
        [0.99999996],
        [1.        ],
        [0.99999996]]))
```

2 Komentář

Obě metody a jejich varianty jsou implementovány dle přednášek. Iterační funkce je identická pro obě metody až na jinou matici Q.

Jednotlivé metody postupně konvergují k výsledku pro gamma = 3 a 2, avšak pro gamma = 1 ani jedna metoda nekonverguje. To je způsobeno tím, že jejich spektrální poloměr (maximum absolutní hodnoty vlastních čísel) matice E - Q^-1 * A je větší než 1, zatímco metoda konverguje právě tehdy když spektrální poloměr je menší než 1.

Zkoušel jsem změnu parametru gama pro obě metody, kde s navyšující hodnotou se pro obě metody výrazně snižoval počet potřebných iterací. Následně jsem sledoval vliv parametru omega u metody GS, kde metoda pro hodnotu 2 a výše nekonverguje, což je s předpokladem s tvrzením 27.4 z přednášky. Tento parametr se používá pro urychlení konvergence a nejnižší počet iterací potřebných k dosažení požadované přesnosti byl naměřen okolo hodnoty 1,2.