Exercise 5 - Mateusz Dorobek

Iterative conjugate Gradient Method with Ax = b, where matrix A is real, symetric and positive defined.

$$\begin{aligned} r_0 &:= b - Ax_0 \\ p_0 &:= r_0 \\ k &:= 0 \end{aligned}$$

repeat

$$\begin{aligned} \alpha_k &:= \frac{r_k^\top r_k}{p_k^\top A p_k} \\ x_{k+1} &:= x_k + \alpha_k p_k \\ r_{k+1} &:= r_k - \alpha_k A p_k \end{aligned}$$

if r is small enough **then** exit loop **end if**

$$\begin{aligned} \beta_k &:= \frac{r_{k+1}^\top r_{k+1}}{r_k^\top r_k} \\ p_{k+1} &:= r_{k+1} + \beta_k p_k \\ k &:= k + 1 \end{aligned}$$

end repeat

Result is:

$$x_{k+1}$$

```python
from IPython.display import display, Math

def bvalue(var, a):
    display(Math(var+' = '+str(round(a,2))))

def bmatrix(var, a):
    """Returns a LaTeX bmatrix

    :a: numpy array
    :returns: LaTeX bmatrix as a string
    """
    if len(a.shape) > 2:
        raise ValueError('bmatrix can at most display two dimensions')
    lines = str(a).replace('[', '').replace(']', '').splitlines()
    rv = [r'\begin{bmatrix}']
    rv += ['  ' + ' & '.join(l.split()) + r'\\' for l in lines]
    rv +=  [r'\end{bmatrix}']
    display(Math(var+' = '+'\n'.join(rv)))
```

```python
from IPython.display import display, Math

def bvalue(var, a):
    return str(var+' = '+str(round(a,2)))
```

```python
def bmatrix(var, a):
    """Returns a LaTeX bmatrix

    :a: numpy array
    :returns: LaTeX bmatrix as a string
    """
    if len(a.shape) > 2:
        raise ValueError('bmatrix can at most display two dimensions')
    lines = str(a).replace('[', '').replace(']', '').splitlines()
    rv = [r'\begin{bmatrix}']
    rv += ['  ' + ' & '.join(l.split()) + r'\\' for l in lines]
    rv +=  [r'\end{bmatrix}']
    return str(var+' = '+'\n'.join(rv))
```

```python
import numpy as np

def conjgrad(A,b,x0):
    r = b - A@x0
    p = r
    x = x0
    disp = ""
    for k, v in {"A": A, "b": b, "x_{0}": x0, "r_{0}": r, "p_{0}": p}.items():
        disp += bmatrix(k,v) + ", "
    display(Math(disp))
    for i in range(A.shape[0]):
        print("Iteration: ", i)
        alpha = (r.T@r)/(p.T@(A@p))
        x = x + alpha*p
        r_new = r - alpha*(A@p)
        if np.linalg.norm(r) < 1e-10:
            break
        beta = (r_new.T@r_new)/(r.T@r)
        p = r_new + beta*p
        r = r_new
        disp = ''
        for k, v in {r"\alpha": alpha.item(), r"\beta": beta.item()}.items():
            disp += bvalue(k,v) + ", "
        for k, v in {"x_{"+str(i+1)+"}": x, "r_{"+str(i+1)+"}": r,
"p_{"+str(i+1)+"}": p}.items():
            disp += bmatrix(k,v) + ", "
        display(Math(disp))
    return x
```

```python
A = np.array([[1, 0, 0], [0, 2, 0], [0, 0, 3]])
b = np.array([[1,1,1]]).T
x0 = np.array([[0,0,0]]).T
x = conjgrad(A,b,x0)
print("Result:")
display(Math(bmatrix("x",x)))
```

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}, b = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, x_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, r_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, p_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix},$$

Iteration: 0

$$\alpha = 0.5, \beta = 0.17, x_1 = \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \end{bmatrix}, r_1 = \begin{bmatrix} 0.5 \\ 0. \\ -0.5 \end{bmatrix}, p_1 = \begin{bmatrix} 0.66666667 \\ 0.16666667 \\ -0.33333333 \end{bmatrix},$$

Iteration: 1

$$\alpha = 0.6, \beta = 0.12, x_2 = \begin{bmatrix} 0.9 \\ 0.6 \\ 0.3 \end{bmatrix}, r_2 = \begin{bmatrix} 0.1 \\ -0.2 \\ 0.1 \end{bmatrix}, p_2 = \begin{bmatrix} 0.18 \\ -0.18 \\ 0.06 \end{bmatrix},$$

Iteration: 2

$$\alpha = 0.56, \beta = 0.0, x_3 = \begin{bmatrix} 1. \\ 0.5 \\ 0.33333333 \end{bmatrix}, r_3 = \begin{bmatrix} 1.38777878e-17 \\ 2.77555756e-17 \\ 1.38777878e-17 \end{bmatrix}, p_3 = \begin{bmatrix} 1.38777878e-17 \\ 2.77555756e-17 \\ 1.38777878e-17 \end{bmatrix},$$

Result:

$$x = \begin{bmatrix} 1. \\ 0.5 \\ 0.33333333 \end{bmatrix}$$

Mateusz Dorobek