Semana8

October 10, 2024

[ ]:
```
%pip install numpy==1.26.4 -q
!python --version

import numpy as np
```

```
Note: you may need to restart the kernel to use updated packages.
Python 3.11.7
```

1 Implementação algoritmica

[ ]:
```
class Newton():

    def __init__(self,
                 tol: float = 10e-6,
                 max_iter: int = 100
                 ) -> None:
        self.tol = tol
        self.max_iter = max_iter
        pass

    def _estimate_jacobian(self,
                           f: 'function',
                           x: np.array,
                           step: float = 1e-6
                           ) -> np.array:

        n = len(x)
        J = np.zeros((n, n))

        for i in range(n):
            x_plus = np.copy(x)
            x_plus[i] += step
            J[:, i] = (f(x_plus) - f(x)) / step
        return J

    def _handle_converged(self,
                          x: np.array,
```

```
                        iter: int) -> None:

        print(100 * '=')
        print(f'Método {self.last_used_method}')
        print(100 * '-')
        print(f'Iterações:')
        print(f'    {iter + 1}')
        print(f'Tolerância: ')
        print(f'    {self.tol}')
        print(f'Resultado: (Arredondado)')
        print(f'    {np.round(x, 3)}')
        print(100 * '=')
        print('')
        return

    def _handle_not_converged(self) -> None:
        raise RuntimeError(
            f'Método {self.last_used_method} não convergiu ' +
            f'mesmo em {self.max_iter + 1} iteraçõos.'
            )

    def solve(self,
              F: 'function',
              x0: np.array
              ) -> np.array:

        self.last_used_method = 'Newton'

        xk = x0
        for k in range(self.max_iter):
            Fk = F(xk)
            Jk = self._estimate_jacobian(F, xk)

            step = np.linalg.solve(Jk, Fk) * (-1)

            xk_new = xk + step

            if (np.linalg.norm(Fk) <= self.tol) \
            or (np.linalg.norm(xk_new - xk) <= self.tol):
                self._handle_converged(xk, k)
                return xk

            xk = xk_new

        self._handle_not_converged()
```

```
    def solve_modified(self,
                       F: 'function',
                       x0: np.array
                       ) -> np.array:

        self.last_used_method = 'Newton Modificado'

        xk = x0
        for k in range(self.max_iter):

            Fk = F(xk)

            if k == 0:
                Jk = self._estimate_jacobian(F, xk)

            step = np.linalg.solve(Jk, Fk) * (-1)

            xk_new = xk + step

            if (np.linalg.norm(Fk) <= self.tol) \
            or (np.linalg.norm(xk_new - xk) <= self.tol):
                self._handle_converged(xk, k)
                return xk

            xk = xk_new

        self._handle_not_converged()
```

1.0.1 Teste - Exemplo 5 do Capítulo 2 (Ruggiero)

[ ]:
```
def F(x: np.array) -> np.array:
    return np.array([
        x[0] + x[1] - 3,
        x[0]**2 + x[1]**2 - 9
    ], dtype=float)

x_0 = np.array([-5.0, 5.0])
```

[ ]:
```
newton = Newton(tol=10e-6, max_iter=100)
newton.solve(F, x_0)
newton.solve_modified(F, x_0)

# Deve resultar em np.array([0, 3])
```

```
================================================================================
====================
Método Newton
--------------------------------------------------------------------------------
```

```
--------------------
Iterações:
    6
Tolerância:
    1e-05
Resultado: (Arredondado)
    [-0.  3.]
================================================================================
====================

================================================================================
====================
Método Newton Modificado
--------------------------------------------------------------------------------
--------------------
Iterações:
    32
Tolerância:
    1e-05
Resultado: (Arredondado)
    [-0.  3.]
================================================================================
====================
```

[ ]:
```
array([-1.83847971e-05,  3.00001838e+00])
```

1.0.2 Exercício 2.A

[ ]:
```
def F(x: np.array) -> np.array:
    return np.array(
        [
            x[0]**2 + x[1]**2 - 2,
            np.exp(x[0] - 1) + x[1]**3 - 2
        ]
    )

epsilon = 10e-4
x0 = np.array([1.5, 2.0])
```

[ ]:
```
newton = Newton(tol=epsilon, max_iter=1000000)
r1 = newton.solve(F, x_0)
r2 = newton.solve_modified(F, x_0)
```

```
================================================================================
====================
Método Newton
--------------------------------------------------------------------------------
```

```
--------------------
Iterações:
    281
Tolerância:
    0.001
Resultado: (Arredondado)
    [-0.714  1.221]
================================================================================
====================


/tmp/ipykernel_152580/1507845510.py:5: RuntimeWarning: overflow encountered in
exp
  np.exp(x[0] - 1) + x[1]**3 - 2
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
Cell In[173], line 3
      1 newton = Newton(tol=epsilon, max_iter=1000000)
      2 r1 = newton.solve(F, x_0)
----> 3 r2 = newton.solve_modified(F, x_0)

Cell In[169], line 106, in Newton.solve_modified(self, F, x0)
    102         return xk
    104     xk = xk_new
--> 106 self._handle_not_converged()

Cell In[169], line 49, in Newton._handle_not_converged(self)
     48 def _handle_not_converged(self) -> None:
---> 49     raise RuntimeError(
     50         f'Método {self.last_used_method} não convergiu ' +
     51         f'mesmo em {self.max_iter + 1} iteraçẽos.'
     52         )

RuntimeError: Método Newton Modificado não convergiu mesmo em 1000001 iteraçẽos
```

O método não é capaz de convergir no Newton modificado, especialmente por conta da sensibilidade da função, que com pequenas variações de x causa grandes impactos em f(x). Assim, manter a mesma equação de convergência em toda a sequência resulta em uma impossibilidade de se atingir um resultado aceitável pelo método modificado.

Por exemplo, reduzindo a tolerância

[ ]:
```
newton = Newton(tol=10e-2, max_iter=100)
r2 = newton.solve_modified(F, x_0)
```

```
================================================================================
====================
```

```
Método Newton Modificado
--------------------------------------------------------------------------------
--------------------
Iterações:
    12
Tolerância:
    0.1
Resultado: (Arredondado)
    [0.625 1.718]
================================================================================
====================
```

é possível encontrar uma resposta para o sistema pelo método. O que pode indicar que o algoritmo tem oscilado ao redor do zero da função, dando passos que não o permitem, quando a tolerância é estrita, atingir um nível de tolerância aceitável.

1.0.3 Exercício 2.B

[ ]:
```python
def F(x: np.array) -> np.array:
    return np.array(
        [
            4 * x[0] - x[0]**3 + x[1],
            - (x[0]**2) / 9 + (4 * x[1] - x[1]**2) / 4 + 1
        ]
    )

epsilon = 10e-4
x0 = np.array([-1, -2], dtype=float)
```

[ ]:
```python
newton = Newton(tol=epsilon, max_iter=100000)
r1 = newton.solve(F, x_0)
r2 = newton.solve_modified(F, x_0)
```

```
================================================================================
====================
Método Newton
--------------------------------------------------------------------------------
--------------------
Iterações:
    21
Tolerância:
    0.001
Resultado: (Arredondado)
    [2.408 4.329]
================================================================================
====================

================================================================================
```

```
====================
Método Newton Modificado
--------------------------------------------------------------------------------
--------------------
Iterações:
    112
Tolerância:
    0.001
Resultado: (Arredondado)
    [2.404 4.332]
================================================================================
====================
```