Rootfinding

Preslav Aleksandrov

September 2020

1 Multivariate Extended Newton

1.1 Original Formulation

The Newton-Raphson method is a well know root finding algorithm which is well defined in the multivariate space. Its vector formulation is as follows:

$$x_i^{n+1} = x_i^n - j_{ik} f_k(x_i^n) \tag{1}$$

Where x_i^{n+1} is the next vector to be used by the method, x_i^n is the current value to be used and j_{ik} is the inverse Jacobian of the function vector $f_k(x_i^n)$. Note that the Jacobian and hence its inverse are square matricies, i.e. $i = k$.

1.2 Extended Formulation

In the extended formulation the Jacobian becomes a rectangular matrix of size $[2n \times n]$, where n stands for the number of equations of the system or in other words the dimension of the system. This new Jacobian matrix will be referenced as the "extended Jacobian" from here on. The inverse of the extended Jacobian will be denoted by p_{ij} which has the transpose shape of the extended Jacobian. The function vector $f_k(x_i^n)$ is extended to align with the shape of p_{ij} to a $[2n \times 1]$ column vector and is written as q_j. The vector x_i^n remains unchanged. These changes can finally be written as:

$$x_i^{n+1} = x_i^n - p_{ij} q_j \tag{2}$$

Where p_{ij} is the pseudo inverse of the extended Jacobian matrix and q_j is the extended column vector of the original function vector.

1.3 Function modification

The essence of the extended Newton method is changing the non-linearity of the original function $r(x)$ [**?**] in order to extend the basin of attraction for that function. The way this is done is by pre-multiplying the function $r(x)$ by a new function called $P(x)$ which is equal to:

$$P(x) = \frac{x - c}{r(x) - r(c)} \tag{3}$$

which transforms the iterable fiction to:

$$f(x) = P(x)r(x) = \frac{x - c}{r(x) - r(c)}r(x) \tag{4}$$

This formulation of $f(x)$ is then expanded into the vector space by changing the input variables to vectors and the main function $r(x)$ to the multivariate space.

$$\begin{aligned} x &\longrightarrow x_i \\ c &\longrightarrow c_i \\ r(x) &\longrightarrow r_k(x_i) \end{aligned} \tag{5}$$

This yields the following equation as the final iterable for the extended newton method:

$$q_j = \overbrace{\left(x_i - c_i\right)}^{A} \overbrace{\frac{r_k(x_i)}{r_k(x_i) - r_k(c_i)}}^{B} \tag{6}$$

Section A becomes a new vector called z_l which is an element wise subtraction of x_i and c_i. Section B becomes a new vector as well, called t_k which is formed by calculating B for each element in r_i and assembling a column vector. This simplifies equation 6 to the following:

$$q_j = z_i t_k, \tag{7}$$

where $j = i \times n + k$. This way of multiplying vectors is also known as the Kronecker product.

1.4 Jacobian assembly

The Jacobian matrix of a vector function is the matrix of all partial first order derivatives. The extended Jacobian of the new modified iterable function q_j is a rectangular matrix denoted by b_{ji}. It is calculated using the following:

$$b_{ji} = \frac{\partial q_j}{\partial x_i} \tag{8}$$

1.5 Pseudo inverse of $b_{ji} \rightarrow p_{ij}$

The pseudo inverse or otherwise known as a Moore–Penrose inverse is a generalisation of the inverse matrix denoted by $\boldsymbol{A}^{+}$. The matrix is commutable and has the transpose shape of the original maintaining a connection with the original terms. A pseudo inverse is a linear map between row and column space. It possess the following properties:

$$\begin{aligned} \boldsymbol{A}\boldsymbol{A}^{+}\boldsymbol{A} &= \boldsymbol{A} \\ \boldsymbol{A}\boldsymbol{A}^{+}\boldsymbol{A} &= \boldsymbol{A} \\ \boldsymbol{A}^{+}\boldsymbol{A}\boldsymbol{A}^{+} &= \boldsymbol{A}^{+} \end{aligned} \tag{9}$$

The most popular way of calculating the pseudo inverse is to use Single Value Decomposition or SVD [**?**]. Multiple software packages offer this feature. In this case NumPy's pinv() function was used. The resulting inverse matrix is denoted by p_{ij}.

1.6 Example Problems [This is where i keep my test functions]

$$
\begin{array}{rcl}
f_1(x) & = & e^{x_1} - x_2 \\
f_2(x) & = & x_1 \times x_2 - e^{x_1} \\
 & & x = (1, 2.718) \\
 & & \\
f_1(x) & = & x_1^2 - x_2^2 - 9 \\
f_2(x) & = & 2x_1 x_2 \\
 & & x = (0, \pm 3) \\
 & & \\
f_1(x) & = & x_1^2 - x_2^3 - x_1 x_2^2 - 1 \\
f_2(x) & = & x_1^3 - x_1 x_2^3 - 4 \\
 & & x = (1.9619, 0.5525) \\
 & & \\
f_1(x) & = & x_1^3 x_2 - x_1 x_2^3 + 6 \\
f_2(x) & = & x_1^2 x_2^2 - x_1 x_2^2 - 8 \\
 & & x = (1, 2) \\
 & & \\
f_1(x) & = & x_1^3 - 3x_1 x_2^2 - 1 \\
f_2(x) & = & 3x_1^2 x_2 - x_2^3 \\
 & & r_1 = (1, 0) \\
 & & r_2 = (-1/2, \sqrt{3}/2) \\
 & & r_3 = (-1/2, -\sqrt{3}/2)
\end{array}
\tag{10}
$$

problems taken from [**?**]

1.7 Comparison

Place holder text

1.7.1 Constant c_i

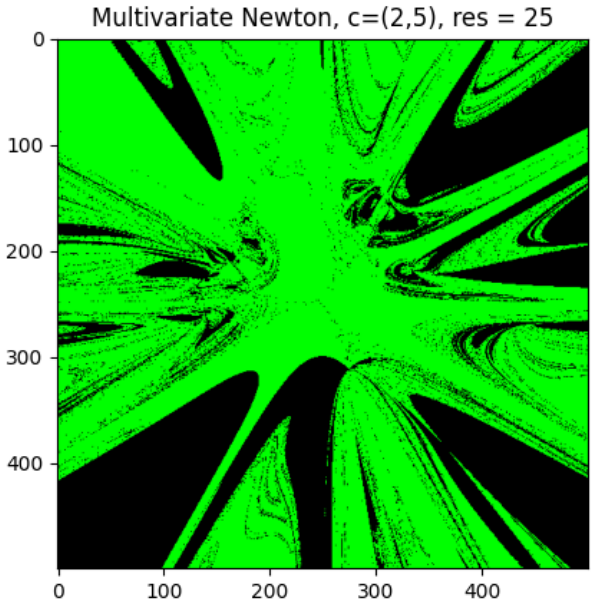

Figure 1: Convergence basin

1.7.2 Constant x_i

2 Broyden's Method

2.1 Comparison

3

unsrt