# Numerical Mathematics Assignment 5 Peter-Jan Derks 11065540

## Exercise 1

The function [zeros]=curveofsolutions(func,derivatives,startingpoint,stepsize,maxsteps , minstepsize,maxstepsize) calculates the set of solutions of a system of n-1 (non-linear) equations with n unknowns. Func, startingpoint and maxsteps are necessary inputs and derivatives,stepsize, minstepsize and maxstepsize are optional inputs. Func should be a column vector of a system of equations. Startingpoint should be a column vector of coordinates and Stepsize and maxsteps should be scalar values representing the size of the first step taken and the maximum number of steps the function should take. Derivatives should be a column vector,if an empty matrix is given the function calculates the derivative itself using the numjac function. Minstepsize and maxstepsize should be positive scalar values, if not provided by the user the function uses values 0 and inf. (The reason I chose these values is that it would otherwise be possible for example that the user gives only a minstepsize that would be higher than the maxstepsize the function chooses if no value is given. If 0 and inf do not work for some function they could be replaced by 0.01 and 10).

In example of an input for the function is:

```
>> curveofsolutions(@(x) x(1)^2+x(2)^2-1, @(x) [2*x(1),2*x(2)], [0.9; 0],0.1,100)
```

This gives output:

```
ans =

  Columns 1 through 11

    0.9950    0.9798    0.9543    0.9184    0.8723    0.8161    0.7503    0.6753    0.5916    0.5002
0.4020
    0.1000    0.2000    0.2989    0.3956    0.4890    0.5778    0.6611    0.7376    0.8062    0.8659
0.9157

….. (Decided not to show all values)
  Columns 89 through 99

   -0.9558   -0.9814   -0.9960   -0.9999   -0.9932   -0.9764   -0.9500   -0.9144   -0.8703   -0.8184   -
0.7595
    0.2940    0.1920    0.0889   -0.0141   -0.1161   -0.2158   -0.3123   -0.4049   -0.4925   -0.5746   -
0.6505
```

## Discription

The functions starts of by checking if the derivative is given. If it isn't given it is calculated using the built in function numjac. To start off the function determines a point close to the startingpoint (given by the user) lying on the curve. Then the function determines in which direction 'v' the curve extends. This is done by taking the null space of the partial derivatives of the function. Then the function takes its first step of size 'stepsize' (given by the user) in the direction of v. This gives point x1=startingpoint+stepsize*v. From this point xq

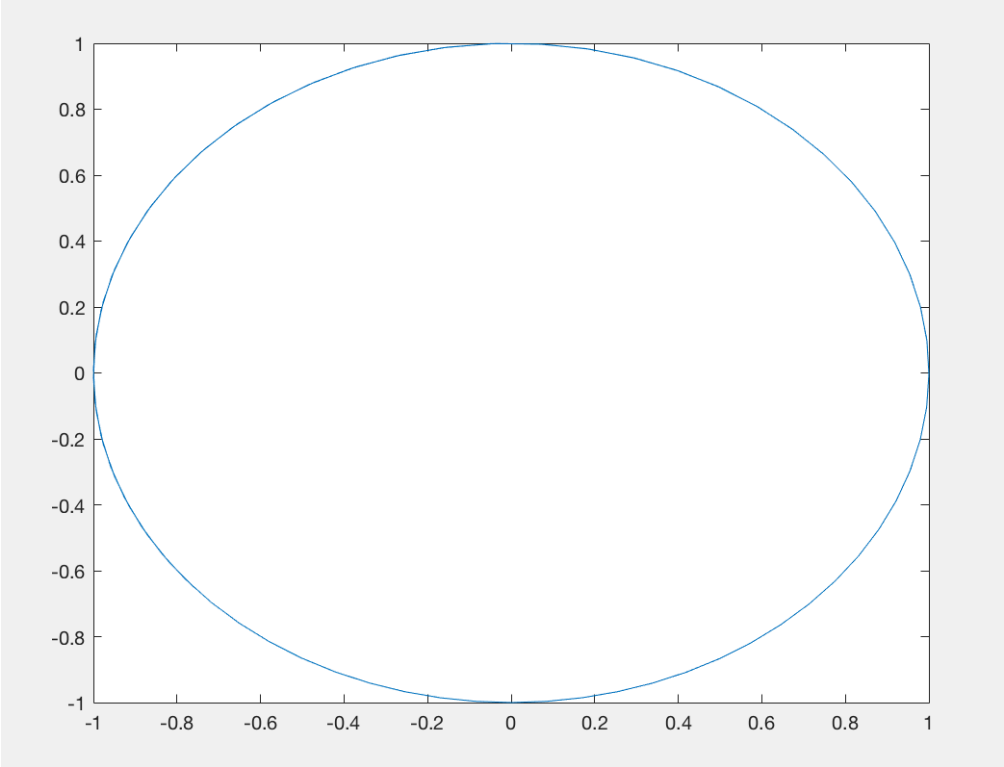
Now finding the direction, using multinewton is repeated N times. Each time it is repeated the function adjusts two variables:

1) The stepsize depending on the distance between the point on v and the closest point lying on the curve
2) The function checks if the angle between subsequent v's is not bigger than 90 degrees. If the angle is bigger than 90 degrees the sign of the stepsize is changed.

The function works for systems of n−1 (non-linear) equations because an extra equation is added to the function and its derivative. To the function $v^T(x_{on\ curve} - x_{on\ v})$ is added and to the derivative $v^T$ is added.

## Picture

The input previously describes produces the following plot:



# Exercise 2

The function PAGERANKINGS=PAGERANK(G,URLS,NUMBEROFSITES) Calculates the pagerank of a set of webpages using a Sparse power method. The input G should be a sparse connection matrix. URLS is an optional input that should be either a row or column vector of urls of webpages connected to the matrix G. NUMBEROFSITES is an optional input that decides the amount of webpages with highest pageranking the user wants to see. If no input for NUMBEROFSITES is given by the user the 10 webpages with highest pageranking are shown. The output PAGERANKINGS is a sorted vector of pagerankings if no urls are given and a table of sorted urls with their corresponding pagerank if urls are given. Additionally, the function produces a bar plot of the pageranks.

# Discription

The function makes use of a sparse power method. The matrix representing the transistion probability matrix of the markov chain can be written as:

$$A = pgD + ez^t$$

The function pagerank computes these elements (p,g,d,e and z) in the following manner: Firstly the column sums of matrix are calculated. This represent the outgoing links. Than a diagonal sparse matrix called D is produced, with values 1/c(k). Than z is produced by implementing the following equation:

$$z_j = \begin{cases} \delta & : \quad c_j \neq 0 \\ 1/n & : \quad c_j = 0. \end{cases}$$

With delta is equal to (1-p)/n. For this function I chose p to be 0.85. P represents the possibility that a random page is clicked.

The precision of the function depends on the tolerance of the while loop. The while stops running if the pagerank values change less than this tolerance. For a very large matrix the tolerance will have to be very low as the total pagerank (equal to 1) needs to be divided over a large number of urls.

In example of an input for the function is:

>> curveofsolutions(@(x) x(1)^2+x(2)^2-1, @(x) [2*x(1),2*x(2)], [0.9; 0],0.1,100)

This gives output:

>> pagerank(sp200000,url200000)

ans =

| pagerank | urls |
|---|---|
| 0.074183 | 'www.wikipedia.org' |
| 0.063061 | 'www.rosso.it' |
| 0.037179 | 'www.rod.sv' |
| 0.017872 | 'www.rod.da' |
| 0.017871 | 'www.vermelho.pt' |
| 0.015197 | 'www.rojo.es' |
| 0.015192 | 'www.rouge.fr' |
| 0.012921 | 'www.rot.de' |
| 0.0063245 | 'www.red.uk' |
| 0.0063238 | 'www.rood.nl' |

Picture

On the x-axis I plotted the size of the sparse matrix input and on the y-axis I plotted the corresponding amount of iterations it took to reach the desired tolerance of the pagerank. It is interesting to see that a larger input does not mean it takes more iterations to calculate a precise pagerank. This has to do with the fact that if the input matrix is bigger the pagerank has to be divided over a larger amount of urls and that thus the tolerance is a smaller percentage of the pagerank per page and is reached with less iterations.