Vectorized zero finders

Prof. Dr. Erhan Coşkun

Karadeniz Technical Univesity,
Department of Mathematics,
E-posta:erhan@ktu.edu.tr

June, 2015

Outline

- Motivation

Outline

- Motivation
- vectorized bisection for real zeros of function in $[-r, r]$

Outline

- Motivation
- vectorized bisection for real zeros of function in $[-r, r]$
- vectorized Newton for all(real and complex) zeros of function in a $B(0, r)$

Outline

- Motivation
- vectorized bisection for real zeros of function in $[-r, r]$
- vectorized Newton for all(real and complex) zeros of function in a $B(0, r)$
- implementations in spectral theory

Outline

- Motivation
- vectorized bisection for real zeros of function in $[-r, r]$
- vectorized Newton for all(real and complex) zeros of function in a $B(0, r)$
- implementations in spectral theory
- vectorized Newton for Nonlinear Systems over a finite domain

Outline

- Motivation
- vectorized bisection for real zeros of function in $[-r, r]$
- vectorized Newton for all(real and complex) zeros of function in a $B(0, r)$
- implementations in spectral theory
- vectorized Newton for Nonlinear Systems over a finite domain
- implementations in plane autonomous systems

Motivation(typical examples)

- Example 1

$$
\begin{aligned}
-u''(x) &= \lambda u(x), 0 < x < L \\
u(0) + au'(0) &= 0 \\
u(L) + bu'(L) &= 0, a > b > 0[DuChateau]
\end{aligned}
$$

Motivation(typical examples)

- Example 1

$$
\begin{aligned}
-u''(x) &= \lambda u(x), 0 < x < L \\
u(0) + au'(0) &= 0 \\
u(L) + bu'(L) &= 0, a > b > 0 [DuChateau]
\end{aligned}
$$

- $\lambda > 0, \lambda_n = k_n^2, k_n \neq 0$, requires

$$
\tan k_n L - \frac{(a-b)k_n}{1 + abk_n^2} = 0, n = 1, 2, \ldots
$$

Motivation(typical examples)

- Example 1

$$\begin{aligned} -u''(x) &= \lambda u(x), 0 < x < L \\ u(0) + au'(0) &= 0 \\ u(L) + bu'(L) &= 0, a > b > 0[DuChateau] \end{aligned}$$

- $\lambda > 0, \lambda_n = k_n^2, k_n \neq 0$, requires

$$\tan k_n L - \frac{(a-b)k_n}{1+abk_n^2} = 0, n = 1, 2, \ldots$$

  - so to generalize, we need to find the zeros of given $f(x)$ over a given interval.

Motivation(typical examples)

- Example 2

$$\int_a^b w(x)f(x)dx \simeq \sum_{i=1}^{n+1} w_i f(x_i)$$

Motivation(typical examples)

- Example 2

$$\int_a^b w(x)f(x)dx \simeq \sum_{i=1}^{n+1} w_i f(x_i)$$

- $\{p_i(x)\}_{i=0}^{n+1}$ are orthogonal with respect to $w(x)$ on $[a, b]$,

Motivation(typical examples)

- Example 2

$$\int_a^b w(x)f(x)dx \simeq \sum_{i=1}^{n+1} w_i f(x_i)$$

- $\{p_i(x)\}_{i=0}^{n+1}$ are orthogonal with respect to $w(x)$ on $[a, b]$,
- x_i, $i = 1, 2, ..., n+1$ are zeros of $p_{n+1}(x)$, (Legendre, Chebyscheff,...)

Motivation(typical examples)

- Example 2

$$\int_a^b w(x)f(x)dx \simeq \sum_{i=1}^{n+1} w_i f(x_i)$$

- $\{p_i(x)\}_{i=0}^{n+1}$ are orthogonal with respect to $w(x)$ on $[a,b]$,
- x_i, $i = 1, 2, ..., n+1$ are zeros of $p_{n+1}(x)$, (Legendre, Chebyscheff,...)
- Furthermore, for oscillatory functions, it is better to divide the range of integration into subintervals with end points corresponding to consecuitive zeros of f to avoid cancellations.[Davis & Rabinowitz]

Motivation(typical examples)

- Example 3

Motivation(typical examples)

- Example 3
- Determine the stationary points of

$$\begin{aligned} \frac{dx}{dt} &= f(x, y) \\ \frac{dy}{dt} &= g(x, y) \end{aligned}$$

over a finite domain, $[a, b] \times [c, d]$

Recall Scalar Bisection Method

- Given a continuous function f defined on $[a, b]$ with $f(a)f(b) < 0$, $\exists c \in (a, b)$ such that $f(c) = 0$.

Recall Scalar Bisection Method

- Given a continuous function f defined on $[a, b]$ with $f(a)f(b) < 0$, $\exists c \in (a, b)$ such that $f(c) = 0$.
- Algorithm

Recall Scalar Bisection Method

- Given a continuous function f defined on $[a, b]$ with $f(a)f(b) < 0$, $\exists c \in (a, b)$ such that $f(c) = 0$.
- Algorithm
  1. input: f, a, b, ϵ

Recall Scalar Bisection Method

- Given a continuous function f defined on $[a, b]$ with $f(a)f(b) < 0$, $\exists c \in (a, b)$ such that $f(c) = 0$.
- Algorithm
  1. input: f, a, b, ϵ
  2. if $f(a)f(b) > 0$ then exit(method does not apply to the problem)

Recall Scalar Bisection Method

- Given a continuous function f defined on $[a, b]$ with $f(a)f(b) < 0$, $\exists c \in (a, b)$ such that $f(c) = 0$.
- Algorithm
  1. input: f, a, b, ϵ
  2. if $f(a)f(b) > 0$ then exit(method does not apply to the problem)
  3. $c = (a + b)/2$

Recall Scalar Bisection Method

- Given a continuous function f defined on $[a, b]$ with $f(a)f(b) < 0$, $\exists c \in (a, b)$ such that $f(c) = 0$.
- Algorithm
  1. input: f, a, b, ϵ
  2. if $f(a)f(b) > 0$ then exit(method does not apply to the problem)
  3. $c = (a + b)/2$
  4. if $f(a)f(c) < 0$ then $b = c$, else $a = c$ (new interval that contains the zero is also called $[a, b]$)

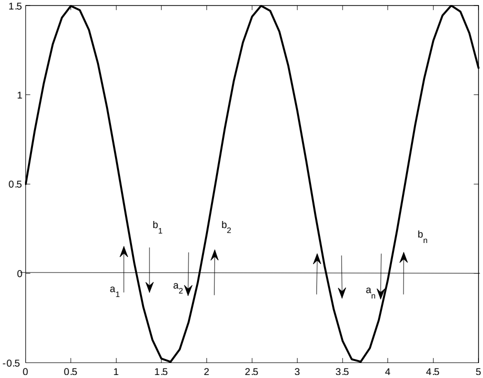
Recall Scalar Bisection Method

- Given a continuous function f defined on $[a, b]$ with $f(a)f(b) < 0$, $\exists c \in (a, b)$ such that $f(c) = 0$.
- Algorithm
  1. input: f, a, b, ϵ
  2. if $f(a)f(b) > 0$ then exit(method does not apply to the problem)
  3. $c = (a + b)/2$
  4. if $f(a)f(c) < 0$ then $b = c$, else $a = c$ (new interval that contains the zero is also called $[a, b]$)
  5. while $|f(c)| > \epsilon$ print $a, c, b, f(c)$ and go to (3); else return c

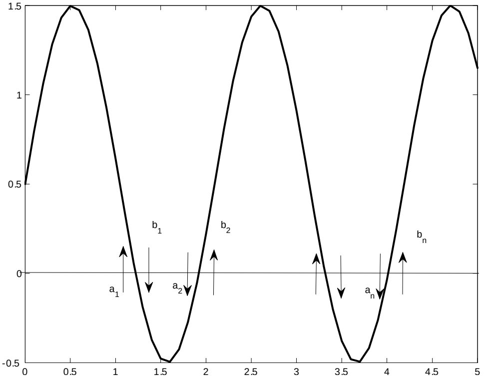
Recall Scalar Bisection Method

- Given a continuous function f defined on $[a, b]$ with $f(a)f(b) < 0$, $\exists c \in (a, b)$ such that $f(c) = 0$.
- Algorithm
  1. input: f, a, b, ϵ
  2. if $f(a)f(b) > 0$ then exit(method does not apply to the problem)
  3. $c = (a + b)/2$
  4. if $f(a)f(c) < 0$ then $b = c$, else $a = c$ (new interval that contains the zero is also called $[a, b]$)
  5. while $|f(c)| > \epsilon$ print $a, c, b, f(c)$ and go to (3); else return c
- determines a single zero! Can we generalize this method to determine simultaneously all zeros of f over $[a, b]$?

Begin with locating intervals on which the function changes sign

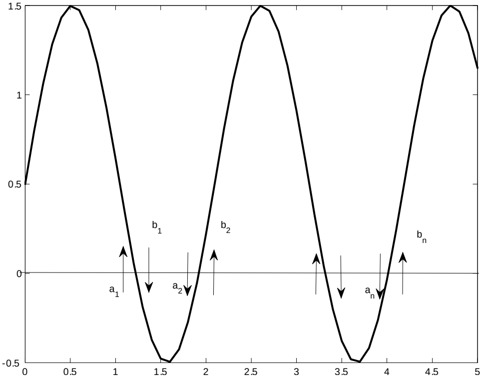


Begin with locating intervals on which the function changes sign

- 

- $A = [a_1, a_2, \ldots, a_n]$; vector of left end points

Begin with locating intervals on which the function changes sign

- 

- $A = [a_1, a_2, \ldots, a_n]$; vector of left end points
- $B = [b_1, b_2, \ldots, b_n]$; vector of right end points

An algorithm to determine the sets A and B, as well as Zeros we may come across.

- bisect_intervals

An algorithm to determine the sets A and B, as well as Zeros we may come across.

- bisect_intervals
  1. input: f, a, b, dx

An algorithm to determine the sets A and B, as well as Zeros we may come across.

- bisect_intervals
  1. input: f, a, b, dx
  2. A: [] B: []

An algorithm to determine the sets A and B, as well as Zeros we may come across.

- bisect_intervals

1. input: f, a, b, dx
2. A: []B: []
3. zeros:[] (to hold zeros that we may come across)

An algorithm to determine the sets A and B, as well as Zeros we may come across.

- bisect_intervals
  1. input: f, a, b, dx
  2. A: []B: []
  3. zeros:[] (to hold zeros that we may come across)
  4. $jump = 1$(allowed numerical variation in $[a, a + dx]$ for a continuous function);

An algorithm to determine the sets A and B, as well as Zeros we may come across.

- bisect_intervals
  1. input: f, a, b, dx
  2. A: $[]B$: $[]$
  3. zeros:[] (to hold zeros that we may come across)
  4. $jump = 1$(allowed numerical variation in $[a, a + dx]$ for a continuous function);
  5. $test$=1(used for stopping criteria)

An algorithm to determine the sets A and B, as well as Zeros we may come across.

- bisect_intervals
  1. input: f, a, b, dx
  2. A: []B: []
  3. zeros:[] (to hold zeros that we may come across)
  4. $jump = 1$(allowed numerical variation in $[a, a + dx]$ for a continuous function);
  5. $test$=1(used for stopping criteria)
  6. while $test = 1$

An algorithm to determine the sets A and B, as well as Zeros we may come across.

- bisect_intervals

1. input: f, a, b, dx
2. A: $[] B$: $[]$
3. zeros:[] (to hold zeros that we may come across)
4. $jump = 1$(allowed numerical variation in $[a, a + dx]$ for a continuous function);
5. $test$=1(used for stopping criteria)
6. while $test = 1$
    - if $f(a) = 0$ then add a to the vector of zeros

An algorithm to determine the sets A and B, as well as Zeros we may come across.

- bisect_intervals

1. input: f, a, b, dx
2. A: $[]B$: $[]$
3. zeros:[] (to hold zeros that we may come across)
4. $jump = 1$(allowed numerical variation in $[a, a + dx]$ for a continuous function);
5. $test$=1(used for stopping criteria)
6. while $test = 1$
   - if $f(a) = 0$ then add a to the vector of zeros
   - if $f(a + dx) = 0$ then add $a + dx$ to the vector of zeros

An algorithm to determine the sets A and B, as well as Zeros we may come across.

- bisect_intervals

1. input: f, a, b, dx
2. A: $[]B$: $[]$
3. zeros:[] (to hold zeros that we may come across)
4. $jump = 1$(allowed numerical variation in $[a, a + dx]$ for a continuous function);
5. $test$=1(used for stopping criteria)
6. while $test = 1$
   - if $f(a) = 0$ then add a to the vector of zeros
   - if $f(a + dx) = 0$ then add $a + dx$ to the vector of zeros
   - if $f(a) * f(a + dx) < 0$ and $abs(f(a) - f(a + dx)) < jump$ then add a to A and $a + dx$ to B

An algorithm to determine the sets A and B, as well as Zeros we may come across.

- bisect_intervals

1. input: f, a, b, dx
2. A: [] B: []
3. zeros:[] (to hold zeros that we may come across)
4. $jump = 1$(allowed numerical variation in $[a, a + dx]$ for a continuous function);
5. $test$=1(used for stopping criteria)
6. while $test = 1$
   - if $f(a) = 0$ then add a to the vector of zeros
   - if $f(a + dx) = 0$ then add $a + dx$ to the vector of zeros
   - if $f(a) * f(a + dx) < 0$ and $abs(f(a) - f(a + dx)) < jump$ then add a to A and $a + dx$ to B
   - $a = a + dx$

An algorithm to determine the sets A and B, as well as Zeros we may come across.

- bisect_intervals

1. input: f, a, b, dx
2. A: $[] B$: $[]$
3. zeros:[] (to hold zeros that we may come across)
4. $jump = 1$(allowed numerical variation in $[a, a + dx]$ for a continuous function);
5. $test$=1(used for stopping criteria)
6. while $test = 1$
   - if $f(a) = 0$ then add a to the vector of zeros
   - if $f(a + dx) = 0$ then add $a + dx$ to the vector of zeros
   - if $f(a) * f(a + dx) < 0$ and $abs(f(a) - f(a + dx)) < jump$ then add a to A and $a + dx$ to B
   - $a = a + dx$
   - if $a >= b$ then $test = 0$

An algorithm to determine the sets A and B, as well as Zeros we may come across.

- bisect_intervals

1. input: f, a, b, dx
2. A: $[] B$: $[]$
3. zeros:[] (to hold zeros that we may come across)
4. $jump = 1$(allowed numerical variation in $[a, a + dx]$ for a continuous function);
5. $test$=1(used for stopping criteria)
6. while $test = 1$
   - if $f(a) = 0$ then add a to the vector of zeros
   - if $f(a + dx) = 0$ then add $a + dx$ to the vector of zeros
   - if $f(a) * f(a + dx) < 0$ and $abs(f(a) - f(a + dx)) < jump$ then add a to A and $a + dx$ to B
   - $a = a + dx$
   - if $a >= b$ then $test = 0$
7. return A and B , as well as vector of zeros.

bisect_intervals

- $f(x) = cos(6cos^{-1}(x))$; (we know the zeros of f)

bisect_intervals

- $f(x) = cos(6cos^{-1}(x))$; (we know the zeros of f)
  - >> [AB,zeros]=bisect_intervals(f,-1,1,0.1)

bisect_intervals

- $f(x) = cos(6cos^{-1}(x))$; (we know the zeros of f)
  - >> [AB,zeros]=bisect_intervals(f,-1,1,0.1)
  - AB =

bisect_intervals

- $f(x) = cos(6cos^{-1}(x))$; (we know the zeros of f)
    - >> [AB,zeros]=bisect_intervals(f,-1,1,0.1)
    - AB =
    - -1.0000 -0.8000 -0.3000 0.2000 0.7000 0.9000

bisect_intervals

- $f(x) = cos(6cos^{-1}(x))$; (we know the zeros of f)
  - >> [AB,zeros]=bisect_intervals(f,-1,1,0.1)
  - AB =
  - -1.0000 -0.8000 -0.3000 0.2000 0.7000 0.9000
  - -0.9000 -0.7000 -0.2000 0.3000 0.8000 1.0000

bisect_intervals

- $f(x) = cos(6cos^{-1}(x))$; (we know the zeros of f)
  - >> [AB,zeros]=bisect_intervals(f,-1,1,0.1)
  - AB =
  - -1.0000 -0.8000 -0.3000 0.2000 0.7000 0.9000
  - -0.9000 -0.7000 -0.2000 0.3000 0.8000 1.0000
  - zeros =

bisect_intervals

- $f(x) = cos(6cos^{-1}(x))$; (we know the zeros of f)
    - >> [AB,zeros]=bisect_intervals(f,-1,1,0.1)
    - AB =
    - -1.0000 -0.8000 -0.3000 0.2000 0.7000 0.9000
    - -0.9000 -0.7000 -0.2000 0.3000 0.8000 1.0000
    - zeros =
    - []

Vectorized Bisection algorithm

- Algorithm

Vectorized Bisection algorithm

- Algorithm
  1. input f, a, b, dx

Vectorized Bisection algorithm

- Algorithm
  1. input f, a, b, dx
  2. call $[AB, zeros] = bisect\_intervals(f, a, b, dx)$ for subintervals AB, zeros

Vectorized Bisection algorithm

- Algorithm
  1. input f, a, b, dx
  2. call $[AB, zeros] = bi\sec t\_intervals(f, a, b, dx)$ for subintervals AB, zeros
  3. set X=$zeros$, $A = AB(:, 1)$; $B = AB(:, 2)$.

Vectorized Bisection algorithm

- Algorithm
  1. input f, a, b, dx
  2. call $[AB, zeros] = bi\sec t\_intervals(f, a, b, dx)$ for subintervals AB, zeros
  3. set X=$zeros$, $A = AB(:,1)$; $B = AB(:,2)$.
  4. if A and X are empty, exit.

Vectorized Bisection algorithm

- Algorithm
  1. input f, a, b, dx
  2. call $[AB, zeros] = bi\sec t\_intervals(f, a, b, dx)$ for subintervals AB, zeros
  3. set X=$zeros$, $A = AB(:,1)$; $B = AB(:,2)$.
  4. if A and X are empty, exit.
  5. if A is empty and X is nonempty, return X.

Vectorized Bisection algorithm

- Algorithm
  1. input f, a, b, dx
  2. call $[AB, zeros] = bi\sec t\_intervals(f, a, b, dx)$ for subintervals AB, zeros
  3. set X=$zeros$, $A = AB(:, 1)$; $B = AB(:, 2)$.
  4. if A and X are empty, exit.
  5. if A is empty and X is nonempty, return X.
  6. set $length$=1, eps=$1e-5$,

Vectorized Bisection algorithm

- Algorithm
  1. input f, a, b, dx
  2. call $[AB, zeros] = bi\sec t\_intervals(f, a, b, dx)$ for subintervals AB, zeros
  3. set X=$zeros$, $A = AB(:, 1)$; $B = AB(:, 2)$.
  4. if A and X are empty, exit.
  5. if A is empty and X is nonempty, return X.
  6. set $length$=1, eps=$1e-5$,
  7. while $||length|| >$ eps do

Vectorized Bisection algorithm

- Algorithm
  1. input f, a, b, dx
  2. call $[AB, zeros] = bi\sec t\_intervals(f, a, b, dx)$ for subintervals AB, zeros
  3. set X=$zeros$, $A = AB(:, 1)$; $B = AB(:, 2)$.
  4. if A and X are empty, exit.
  5. if A is empty and X is nonempty, return X.
  6. set $length$=1, eps=$1e-5$,
  7. while $||length|| > eps$ do
     - $C = (A + B)/2$, vector of midpoints

Vectorized Bisection algorithm

- Algorithm
  1. input f, a, b, dx
  2. call $[AB, zeros] = bi\sec t\_intervals(f, a, b, dx)$ for subintervals AB, zeros
  3. set X=$zeros$, $A = AB(:, 1)$; $B = AB(:, 2)$.
  4. if A and X are empty, exit.
  5. if A is empty and X is nonempty, return X.
  6. set $length$=1, eps=$1e-5$,
  7. while $||length|| >$ eps do
     - $C = (A + B)/2$, vector of midpoints
     - determine the set of indices ii for which $f(A). * f(C) < 0$

Vectorized Bisection algorithm

- Algorithm
  1. input f, a, b, dx
  2. call $[AB, zeros] = bisect\_intervals(f, a, b, dx)$ for subintervals AB, zeros
  3. set X=$zeros$, $A = AB(:,1)$; $B = AB(:,2)$.
  4. if A and X are empty, exit.
  5. if A is empty and X is nonempty, return X.
  6. set $length$=1, eps=$1e-5$,
  7. while $||length|| >$ eps do
     - $C = (A+B)/2$, vector of midpoints
     - determine the set of indices ii for which $f(A).*f(C) < 0$
     - if the set $ii \neq \emptyset$,set $B(ii) = C(ii)$.

Vectorized Bisection algorithm

- Algorithm
  1. input f, a, b, dx
  2. call $[AB, zeros] = bi\sec t\_intervals(f, a, b, dx)$ for subintervals AB, zeros
  3. set X=$zeros$, $A = AB(:,1)$; $B = AB(:,2)$.
  4. if A and X are empty, exit.
  5. if A is empty and X is nonempty, return X.
  6. set $length$=1, eps=$1e-5$,
  7. while $||length|| >$ eps do
     - $C = (A+B)/2$, vector of midpoints
     - determine the set of indices ii for which $f(A).*f(C) < 0$
     - if the set $ii \neq \varnothing$,set $B(ii) = C(ii)$.
     - determine the set of indices jj for which $f(A).*f(C) >= 0$.

Vectorized Bisection algorithm

- Algorithm
  1. input f, a, b, dx
  2. call $[AB, zeros] = bisect\_intervals(f, a, b, dx)$ for subintervals AB, zeros
  3. set X=$zeros$, $A = AB(:,1)$; $B = AB(:,2)$.
  4. if A and X are empty, exit.
  5. if A is empty and X is nonempty, return X.
  6. set $length$=1, eps=$1e-5$,
  7. while $||length|| >$ eps do
     - $C = (A + B)/2$, vector of midpoints
     - determine the set of indices ii for which $f(A). * f(C) < 0$
     - if the set $ii \neq \emptyset$,set $B(ii) = C(ii)$.
     - determine the set of indices jj for which $f(A). * f(C) >= 0$.
     - if the set $jj \neq \emptyset$, set $A(jj) = C(jj)$.

Vectorized Bisection algorithm

- Algorithm
  1. input f, a, b, dx
  2. call $[AB, zeros] = bisect\_intervals(f, a, b, dx)$ for subintervals AB, zeros
  3. set X=$zeros$, $A = AB(:,1)$; $B = AB(:,2)$.
  4. if A and X are empty, exit.
  5. if A is empty and X is nonempty, return X.
  6. set $length$=1, eps=$1e-5$,
  7. while $||length|| >$ eps do
     - $C = (A + B)/2$, vector of midpoints
     - determine the set of indices ii for which $f(A). * f(C) < 0$
     - if the set $ii \neq \emptyset$,set $B(ii) = C(ii)$.
     - determine the set of indices jj for which $f(A). * f(C) >= 0$.
     - if the set $jj \neq \emptyset$, set $A(jj) = C(jj)$.
     - set $length = ||B - A||$

Vectorized Bisection algorithm

- Algorithm
  1. input f, a, b, dx
  2. call $[AB, zeros] = bi\sec t\_intervals(f, a, b, dx)$ for subintervals AB, zeros
  3. set $X{=}zeros$, $A = AB(:,1)$; $B = AB(:,2)$.
  4. if A and X are empty, exit.
  5. if A is empty and X is nonempty, return X.
  6. set $length{=}1$, $eps{=}1e-5$,
  7. while $||length|| >$ eps do
     - $C = (A+B)/2$, vector of midpoints
     - determine the set of indices ii for which $f(A).*f(C) < 0$
     - if the set $ii \neq \varnothing$,set $B(ii) = C(ii)$.
     - determine the set of indices jj for which $f(A).*f(C) >= 0$.
     - if the set $jj \neq \varnothing$, set $A(jj) = C(jj)$.
     - set $length = ||B - A||$
     - determine the indices $j0$ for which $length <= eps$ or $||f(C)|| < eps$

Vectorized Bisection algorithm

- Algorithm
  1. input f, a, b, dx
  2. call $[AB, zeros] = bi\sec t\_intervals(f, a, b, dx)$ for subintervals AB, zeros
  3. set X=$zeros$, $A = AB(:, 1)$; $B = AB(:, 2)$.
  4. if A and X are empty, exit.
  5. if A is empty and X is nonempty, return X.
  6. set $length$=1, eps=$1e-5$,
  7. while $||length|| >$ eps do
     - $C = (A + B)/2$, vector of midpoints
     - determine the set of indices ii for which $f(A). * f(C) < 0$
     - if the set $ii \neq \emptyset$,set $B(ii) = C(ii)$.
     - determine the set of indices jj for which $f(A). * f(C) >= 0$.
     - if the set $jj \neq \emptyset$, set $A(jj) = C(jj)$.
     - set $length = ||B - A||$
     - determine the indices $j0$ for which $length <= eps$ or $||f(C)|| < eps$
     - add $C(j0)$ to X, i.e., $X = [X; C(j0)]$.

Vectorized Bisection algorithm

- Algorithm
  1. input f, a, b, dx
  2. call $[AB, zeros] = bi\sec t\_intervals(f, a, b, dx)$ for subintervals AB, zeros
  3. set X=$zeros$, $A = AB(:,1)$; $B = AB(:,2)$.
  4. if A and X are empty, exit.
  5. if A is empty and X is nonempty, return X.
  6. set $length$=1, eps=$1e-5$,
  7. while $||length|| >$ eps do
     - $C = (A + B)/2$, vector of midpoints
     - determine the set of indices ii for which $f(A). * f(C) < 0$
     - if the set $ii \neq \varnothing$,set $B(ii) = C(ii)$.
     - determine the set of indices jj for which $f(A). * f(C) >= 0$.
     - if the set $jj \neq \varnothing$, set $A(jj) = C(jj)$.
     - set $length = ||B - A||$
     - determine the indices $j0$ for which $length <= eps$ or $||f(C)|| < eps$
     - add $C(j0)$ to X, i.e., $X = [X; C(j0)]$.
     - determine the indices $j1$ for which $length > eps$

Vectorized Bisection algorithm

- Algorithm
  1. input f, a, b, dx
  2. call $[AB, zeros] = bi\sec t\_intervals(f, a, b, dx)$ for subintervals AB, zeros
  3. set X=$zeros$, $A = AB(:, 1)$; $B = AB(:, 2)$.
  4. if A and X are empty, exit.
  5. if A is empty and X is nonempty, return X.
  6. set $length$=1, eps=$1e-5$,
  7. while $||length|| >$ eps do
     - $C = (A + B)/2$, vector of midpoints
     - determine the set of indices ii for which $f(A). * f(C) < 0$
     - if the set $ii \neq \emptyset$,set $B(ii) = C(ii)$.
     - determine the set of indices jj for which $f(A). * f(C) >= 0$.
     - if the set $jj \neq \emptyset$, set $A(jj) = C(jj)$.
     - set $length = ||B - A||$
     - determine the indices $j0$ for which $length <= eps$ or $||f(C)|| < eps$
     - add $C(j0)$ to X, i.e., $X = [X; C(j0)]$.
     - determine the indices $j1$ for which $length > eps$
     - if $j1 \neq \emptyset$ then set $A = A(j1)$ ve $B = B(j1)$.

Vectorized Bisection algorithm

- Algorithm
  1. input f, a, b, dx
  2. call $[AB, zeros] = bi\sec t\_intervals(f, a, b, dx)$ for subintervals AB, zeros
  3. set $X{=}zeros$, $A = AB(:,1)$; $B = AB(:,2)$.
  4. if A and X are empty, exit.
  5. if A is empty and X is nonempty, return X.
  6. set $length{=}1$, $eps{=}1e-5$,
  7. while $||length|| >$ eps do
     - $C = (A+B)/2$, vector of midpoints
     - determine the set of indices ii for which $f(A).*f(C) < 0$
     - if the set $ii \neq \varnothing$,set $B(ii) = C(ii)$.
     - determine the set of indices jj for which $f(A).*f(C) >= 0$.
     - if the set $jj \neq \varnothing$, set $A(jj) = C(jj)$.
     - set $length = ||B - A||$
     - determine the indices $j0$ for which $length <= eps$ or $||f(C)|| < eps$
     - add $C(j0)$ to X, i.e., $X = [X; C(j0)]$.
     - determine the indices $j1$ for which $length > eps$
     - if $j1 \neq \varnothing$ then set $A = A(j1)$ ve $B = B(j1)$.
  8. return X

Vectorized Bisection

- Simple tests

Vectorized Bisection

- Simple tests
- $>> f(x) = cos(6cos^{-1}(x));$

Vectorized Bisection

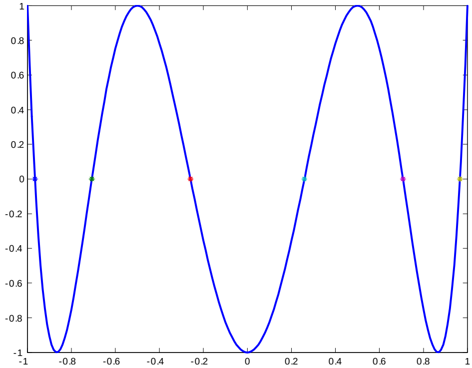
- Simple tests
- $>> f(x) = cos(6cos^{-1}(x));$
- $>>$ X=bisectv(f,-1,1,0.1)

Vectorized Bisection

- Simple tests
- $>> f(x) = cos(6cos^{-1}(x));$
- $>>$ X=bisectv(f,-1,1,0.1)
- X =-0.9659 -0.7071 -0.2588 0.2588 0.7071 0.9659

Vectorized Bisection

- Simple tests
- $>> f(x) = cos(6cos^{-1}(x));$
- >> X=bisectv(f,-1,1,0.1)
- X =-0.9659 -0.7071 -0.2588 0.2588 0.7071 0.9659



Vectorized Bisection

- f(x)=tan(x)-x;

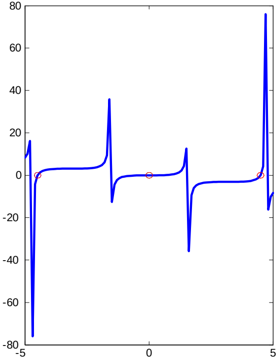
Vectorized Bisection

- f(x)=tan(x)-x;
- >> X=bisectv(f,-5,5,0.1)

Vectorized Bisection

- f(x)=tan(x)-x;
- $>>$ X=bisectv(f,-5,5,0.1)
- X = -4.4934 0.0000 4.4934

Vectorized Bisection

- f(x)=tan(x)-x;
- >> X=bisectv(f,-5,5,0.1)
- X = -4.4934 0.0000 4.4934
- 

Vectorized Bisection(applications to spectral theory)

- $$\begin{aligned} -u''(x) &= \lambda u(x), 0 < x < L \\ u(0) + au'(0) &= 0 \\ u(L) + bu'(L) &= 0, a > b > 0[DuChateau] \end{aligned}$$

Vectorized Bisection(applications to spectral theory)

- $$\begin{aligned} -u''(x) &= \lambda u(x), 0 < x < L \\ u(0) + au'(0) &= 0 \\ u(L) + bu'(L) &= 0, a > b > 0[DuChateau] \end{aligned}$$

- $\lambda > 0, \lambda_n = k_n^2, k_n \neq 0$, requires

$$\tan k_n L = \frac{(a-b)k_n}{1 + abk_n^2}, n = 1, 2, \ldots$$

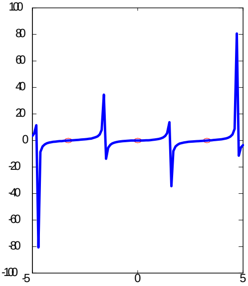
Vectorized Bisection(applications to spectral theory)

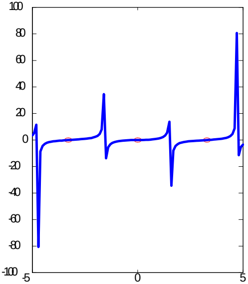
- L=1;a=2,b=1;

Vectorized Bisection(applications to spectral theory)

- L=1;a=2,b=1;
- $f(x) = tan(x) - x./(1 + 2x.\hat{}2)$

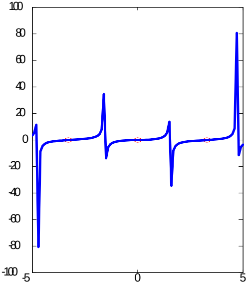
Vectorized Bisection(applications to spectral theory)

- L=1;a=2,b=1;
- $f(x) = tan(x) - x./(1 + 2x.\hat{}2)$
- 

- L=1;a=2,b=1;
- $f(x) = tan(x) - x./(1 + 2x.\hat{}2)$
- 

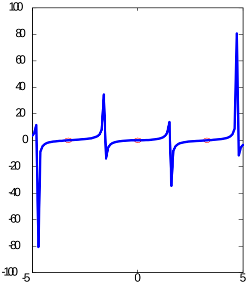
- >> X=bisectv(f,-5,5,0.1)

Vectorized Bisection(applications to spectral theory)

- L=1;a=2,b=1;
- $f(x) = tan(x) - x./(1 + 2x.\hat{}2)$
- 

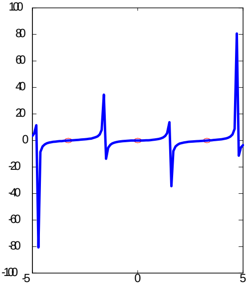
- $>>$ X=bisectv(f,-5,5,0.1)
- X =-3.2860 0.0000 3.2860

Vectorized Bisection(applications to spectral theory)

- L=1;a=2,b=1;
- $f(x) = tan(x) - x./(1 + 2x.\hat{}2)$
- 

- >> X=bisectv(f,-5,5,0.1)
- X =-3.2860   0.0000   3.2860
- Notice the points of discontinuities;

Vectorized Bisection(applications to spectral theory)

- L=1;a=2,b=1;
- $f(x) = tan(x) - x./(1 + 2x.\hat{}2)$
- 

- $>>$ X=bisectv(f,-5,5,0.1)
- X =-3.2860 0.0000 3.2860
- Notice the points of discontinuities;
- fzero(f,1), MATLAB ,ans =1.5708,$>>$ f(ans),ans =-1.2093e+015(wrong!)

Vectorized Newton(determines all zeros of a function in B(0,r))

- Algorithm

Vectorized Newton(determines all zeros of a function in B(0,r))

- Algorithm
  1. input $f, df, r, dx, zerotype$

Vectorized Newton(determines all zeros of a function in B(0,r))

- Algorithm
  1. input f, df, r, dx, $zerotype$
  2. $zerotype = 0$(real zeros only),$zerotype = 1$, all zeros

Vectorized Newton(determines all zeros of a function in B(0,r))

- Algorithm
  1. input $f, df, r, dx, zerotype$
  2. $zerotype = 0$(real zeros only),$zerotype = 1$, all zeros
  3. i$fzerotype = 0$, set $Z0 = -r : dx : r$;

Vectorized Newton(determines all zeros of a function in B(0,r))

- Algorithm
  1. input $f, df, r, dx, zerotype$
  2. $zerotype = 0$(real zeros only),$zerotype = 1$, all zeros
  3. i$fzerotype = 0$, set $Z0 = -r : dx : r$;
  4. else set $x = -r : dx : r$, $y = x$; $Z = X + i * Y$;matrix of initial values

Vectorized Newton(determines all zeros of a function in B(0,r))

- Algorithm
  1. input $f, df, r, dx, zerotype$
  2. $zerotype = 0$(real zeros only),$zerotype = 1$, all zeros
  3. i$fzerotype = 0$, set $Z0 = -r : dx : r$;
  4. else set $x = -r : dx : r$, $y = x; Z = X + i * Y$;matrix of initial values
     - $Z0 = vectorize(Z)$; vector of initial values

Vectorized Newton(determines all zeros of a function in B(0,r))

- Algorithm
  1. input $f, df, r, dx, zerotype$
  2. $zerotype = 0$(real zeros only),$zerotype = 1$, all zeros
  3. i$fzerotype = 0$, set $Z0 = -r : dx : r$;
  4. else set $x = -r : dx : r$, $y = x$; $Z = X + i * Y$;matrix of initial values
     - $Z0 = vectorize(Z)$; vector of initial values
  5. until convergence do

Vectorized Newton(determines all zeros of a function in B(0,r))

- Algorithm
  1. input $f, df, r, dx, zerotype$
  2. $zerotype = 0$(real zeros only),$zerotype = 1$, all zeros
  3. i$fzerotype = 0$, set $Z0 = -r : dx : r$;
  4. else set $x = -r : dx : r$, $y = x; Z = X + i * Y$;matrix of initial values
     - $Z0 = vectorize(Z)$; vector of initial values
  5. until convergence do
     - $Z1 = Z0 - f(Z0)./df(Z0)$;

Vectorized Newton(determines all zeros of a function in B(0,r))

- Algorithm
  1. input $f, df, r, dx, zerotype$
  2. $zerotype = 0$(real zeros only),$zerotype = 1$, all zeros
  3. i$fzerotype = 0$, set $Z0 = -r : dx : r$;
  4. else set $x = -r : dx : r$, $y = x$; $Z = X + i * Y$;matrix of initial values
     - $Z0 = vectorize(Z)$; vector of initial values
  5. until convergence do
     - $Z1 = Z0 - f(Z0)./df(Z0)$;
     - difference=$abs(Z1 - Z0)$;

Vectorized Newton(determines all zeros of a function in B(0,r))

- Algorithm
  1. input $f, df, r, dx, zerotype$
  2. $zerotype = 0$(real zeros only),$zerotype = 1$, all zeros
  3. i$fzerotype = 0$, set $Z0 = -r : dx : r$;
  4. else set $x = -r : dx : r$, $y = x; Z = X + i * Y$;matrix of initial values
     - $Z0 = vectorize(Z)$; vector of initial values
  5. until convergence do
     - $Z1 = Z0 - f(Z0)./df(Z0)$;
     - difference=$abs(Z1 - Z0)$;
     - $j0$ =find($difference <= eps$)

Vectorized Newton(determines all zeros of a function in B(0,r))

- Algorithm
  1. input $f, df, r, dx, zerotype$
  2. $zerotype = 0$(real zeros only),$zerotype = 1$, all zeros
  3. i$fzerotype = 0$, set $Z0 = -r : dx : r$;
  4. else set $x = -r : dx : r$, $y = x$; $Z = X + i * Y$;matrix of initial values
     - $Z0 = vectorize(Z)$; vector of initial values
  5. until convergence do
     - $Z1 = Z0 - f(Z0)./df(Z0)$;
     - difference=$abs(Z1 - Z0)$;
     - $j0$ =find($difference <= eps$)
     - $j1$ =find($(difference > eps)\&abs(Z1) < r$);

Vectorized Newton(determines all zeros of a function in B(0,r))

- Algorithm
  1. input $f, df, r, dx, zerotype$
  2. $zerotype = 0$(real zeros only),$zerotype = 1$, all zeros
  3. i$fzerotype = 0$, set $Z0 = -r : dx : r$;
  4. else set $x = -r : dx : r$, $y = x$; $Z = X + i * Y$;matrix of initial values
     - $Z0 = vectorize(Z)$; vector of initial values
  5. until convergence do
     - $Z1 = Z0 - f(Z0)./df(Z0)$;
     - difference=$abs(Z1 - Z0)$;
     - $j0$ =find($difference <= eps$)
     - $j1$ =find(($difference > eps$)&$abs(Z1) < r$);
     - $Z = [Z; Z1(j0)]$; converged components

Vectorized Newton(determines all zeros of a function in B(0,r))

- Algorithm
  1. input $f, df, r, dx, zerotype$
  2. $zerotype = 0$(real zeros only),$zerotype = 1$, all zeros
  3. i$fzerotype = 0$, set $Z0 = -r : dx : r$;
  4. else set $x = -r : dx : r$, $y = x; Z = X + i * Y$;matrix of initial values
     - $Z0 = vectorize(Z)$; vector of initial values
  5. until convergence do
     - $Z1 = Z0 - f(Z0)./df(Z0)$;
     - difference$=abs(Z1 - Z0)$;
     - $j0$ =find$(difference <= eps)$
     - $j1$ =find$((difference > eps)\&abs(Z1) < r)$;
     - $Z = [Z; Z1(j0)]$; converged components
     - $Z0 = Z1(j1)$; continue with components yet to converge

Vectorized Newton(determines all zeros of a function in B(0,r))

- Algorithm
  1. input $f, df, r, dx, zerotype$
  2. $zerotype = 0$(real zeros only),$zerotype = 1$, all zeros
  3. i$fzerotype = 0$, set $Z0 = -r : dx : r$;
  4. else set $x = -r : dx : r$, $y = x$; $Z = X + i * Y$;matrix of initial values
     - $Z0 = vectorize(Z)$; vector of initial values
  5. until convergence do
     - $Z1 = Z0 - f(Z0)./df(Z0)$;
     - difference=$abs(Z1 - Z0)$;
     - $j0$ =find($difference <= eps$)
     - $j1$ =find(($difference > eps$)&$abs(Z1) < r$);
     - $Z = [Z; Z1(j0)]$; converged components
     - $Z0 = Z1(j1)$; continue with components yet to converge
  6. return nonrepeating Z values in$B(0, r)$

Vectorized Newton(determines all zeros of a function in B(0,r))

- Algorithm
  1. input $f, df, r, dx, zerotype$
  2. $zerotype = 0$(real zeros only),$zerotype = 1$, all zeros
  3. i$fzerotype = 0$, set $Z0 = -r : dx : r$;
  4. else set $x = -r : dx : r$, $y = x$; $Z = X + i * Y$;matrix of initial values
     - $Z0 = vectorize(Z)$; vector of initial values
  5. until convergence do
     - $Z1 = Z0 - f(Z0)./df(Z0)$;
     - difference=$abs(Z1 - Z0)$;
     - $j0$ =find$(difference <= eps)$
     - $j1$ =find$((difference > eps)\&abs(Z1) < r)$;
     - $Z = [Z; Z1(j0)]$; converged components
     - $Z0 = Z1(j1)$; continue with components yet to converge
  6. return nonrepeating Z values in$B(0, r)$
- The case of real zeros has been further investigated in Memoglu[MS thesis].

- $f(x) = x^4 + x^3 + x^2 + x + 1$

- $f(x) = x^4 + x^3 + x^2 + x + 1$
- $df(x) = 4x^3 + 3x^2 + 2x + 1$

- $f(x) = x^4 + x^3 + x^2 + x + 1$
- $df(x) = 4x^3 + 3x^2 + 2x + 1$

```
>> cvnewton(f,df,2,0.1,1)

ans =

   0.3090 - 0.9511i
   0.3090 + 0.9511i
  -0.8090 - 0.5878i
  -0.8090 + 0.5878i
```

```
>> roots([1 1 1 1 1])

ans =

   0.3090 + 0.9511i
   0.3090 - 0.9511i
  -0.8090 + 0.5878i
  -0.8090 - 0.5878i
```

Vectorized Newton:all zeros of a function in B(0,r)

- $f(z) = \sin(z\hat{}2 + 1)$

Vectorized Newton:all zeros of a function in B(0,r)

- $f(z) = \sin(z\hat{}2 + 1)$
- r=3

Vectorized Newton:all zeros of a function in B(0,r)

- $f(z) = \sin(z\hat{}2 + 1)$
- r=3

z	$\sin(z^2 + 1)$
	1.0e-007 *
0 - 1.0000i	0
0 + 1.0000i	0
1.4634	0.0111
-1.4634	0.0111
0 - 2.0351i	-0.0233
0 + 2.0351i	-0.0233
2.2985	0.1094
-2.2985	0.1094
0 - 2.6987i	0.2583
0 + 2.6987i	0.2583
0 - 2.6987i	-0.2815
0 + 2.6987i	-0.2815
2.9025	-0.2347
-2.9025	-0.2347

Vectorized Newton:all zeros of a function in B(0,r)

- $f(z) = \sin(e^z)$

Vectorized Newton:all zeros of a function in B(0,r)

- $f(z) = \sin(e^z)$
- r=3.5

Vectorized Newton:all zeros of a function in B(0,r)

- $f(z) = \sin(e^z)$
- r=3.5

z	f(z)
	1.0e-003 *
1.1447	-0.0004
1.8379	0.0184
2.2433	0.0205
2.5310	-0.0534
2.7542	-0.0346
2.9365	0.0122
3.0906	0.0008
3.2242	-0.0359
3.3420	0.1262
1.1447 - 3.1416i	0.0004 + 0.0083i
1.1447 + 3.1416i	0.0004 - 0.0083i
3.4473	-0.1564

Vectorized Newton:all zeros of a function in B(0,r)

- $f(z) = \cos(2z)$

Vectorized Newton:all zeros of a function in B(0,r)

- $f(z) = \cos(2z)$
- r=4

Vectorized Newton:all zeros of a function in B(0,r)

- $f(z) = \cos(2z)$
- r=4

z	f(z)
	1.0e-005 *
-3.9270	0.1634
-2.3562	-0.8980
-0.7854	-0.3673
0.7854	-0.3673
2.3562	-0.8980
3.9270	0.1634

Vectorized Newton for nonlinear systems

- First consider the conventional Newton for the nonlinear system

Vectorized Newton for nonlinear systems

- First consider the conventional Newton for the nonlinear system
-

$$\begin{aligned} f(x,y) &= 0 \\ g(x,y) &= 0 \end{aligned}$$

Vectorized Newton for nonlinear systems

- First consider the conventional Newton for the nonlinear system
-

$$\begin{aligned} f(x,y) &= 0 \\ g(x,y) &= 0 \end{aligned}$$

1. Choose $X^{(0)} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$,

Vectorized Newton for nonlinear systems

- First consider the conventional Newton for the nonlinear system
-

$$
\begin{aligned}
f(x, y) &= 0 \\
g(x, y) &= 0
\end{aligned}
$$

1. Choose $X^{(0)} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$,
2. for $i = 0$ until convergence do

Vectorized Newton for nonlinear systems

- First consider the conventional Newton for the nonlinear system
-

$$
\begin{aligned}
f(x,y) &= 0 \\
g(x,y) &= 0
\end{aligned}
$$

1. Choose $X^{(0)} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$,
2. for $i = 0$ until convergence do
    1. $J(X^{(i)}) = \begin{bmatrix} f_x & f_y \\ g_x & g_y \end{bmatrix}_{(X^{(i)})}, F(X^{(i)}) = \begin{bmatrix} f(x_i, y_i) \\ g(x_i, y_i) \end{bmatrix},$

Vectorized Newton for nonlinear systems

- First consider the conventional Newton for the nonlinear system
-

$$\begin{aligned} f(x,y) &= 0 \\ g(x,y) &= 0 \end{aligned}$$

1. Choose $X^{(0)} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$,
2. for $i = 0$ until convergence do
    1. $J(X^{(i)}) = \begin{bmatrix} f_x & f_y \\ g_x & g_y \end{bmatrix}_{(X^{(i)})}$, $F(X^{(i)}) = \begin{bmatrix} f(x_i, y_i) \\ g(x_i, y_i) \end{bmatrix}$,
    2. Solve $J(X^{(i)})\Delta X = -F(X^{(i)})$

Vectorized Newton for nonlinear systems

- First consider the conventional Newton for the nonlinear system
- $$\begin{aligned} f(x,y) &= 0 \\ g(x,y) &= 0 \end{aligned}$$

  1. Choose $X^{(0)} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$,
  2. for $i = 0$ until convergence do
     1. $J(X^{(i)}) = \begin{bmatrix} f_x & f_y \\ g_x & g_y \end{bmatrix}_{(X^{(i)})}$, $F(X^{(i)}) = \begin{bmatrix} f(x_i, y_i) \\ g(x_i, y_i) \end{bmatrix}$,
     2. Solve $J(X^{(i)})\Delta X = -F(X^{(i)})$
     3. $X^{(i+1)} = X^{(i)} + \Delta X$

Vectorized Newton for nonlinear systems

- First consider the conventional Newton for the nonlinear system
-

$$\begin{aligned} f(x,y) &= 0 \\ g(x,y) &= 0 \end{aligned}$$

1. Choose $X^{(0)} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$,
2. for $i = 0$ until convergence do
    1. $J(X^{(i)}) = \begin{bmatrix} f_x & f_y \\ g_x & g_y \end{bmatrix}_{(X^{(i)})}$, $F(X^{(i)}) = \begin{bmatrix} f(x_i, y_i) \\ g(x_i, y_i) \end{bmatrix}$,
    2. Solve $J(X^{(i)})\Delta X = -F(X^{(i)})$
    3. $X^{(i+1)} = X^{(i)} + \Delta X$

- it works for a "good" choice of $X^{(0)}$

Vectorized Newton for nonlinear systems

- First consider the conventional Newton for the nonlinear system
- $$\begin{aligned} f(x,y) &= 0 \\ g(x,y) &= 0 \end{aligned}$$

1. Choose $X^{(0)} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$,
2. for $i = 0$ until convergence do
    1. $J(X^{(i)}) = \begin{bmatrix} f_x & f_y \\ g_x & g_y \end{bmatrix}_{(X^{(i)})}$, $F(X^{(i)}) = \begin{bmatrix} f(x_i, y_i) \\ g(x_i, y_i) \end{bmatrix}$,
    2. Solve $J(X^{(i)})\Delta X = -F(X^{(i)})$
    3. $X^{(i+1)} = X^{(i)} + \Delta X$

- it works for a "good" choice of $X^{(0)}$
- provided that $J(X^{(i)})$ is nonsingular

Vectorized Newton for nonlinear systems

- First consider the conventional Newton for the nonlinear system
-

$$
\begin{aligned}
f(x,y) &= 0 \\
g(x,y) &= 0
\end{aligned}
$$

1. Choose $X^{(0)} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$,
2. for $i = 0$ until convergence do
    1. $J(X^{(i)}) = \begin{bmatrix} f_x & f_y \\ g_x & g_y \end{bmatrix}_{(X^{(i)})}$, $F(X^{(i)}) = \begin{bmatrix} f(x_i, y_i) \\ g(x_i, y_i) \end{bmatrix}$,
    2. Solve $J(X^{(i)})\Delta X = -F(X^{(i)})$
    3. $X^{(i+1)} = X^{(i)} + \Delta X$

- it works for a "good" choice of $X^{(0)}$
- provided that $J(X^{(i)})$ is nonsingular
- determines a single solution

Vectorized Newton for nonlinear systems:all zeros over

$[a, b]x[c, d]$

- Can we find all zeros(say real, for example) of a nonlinear system in $[a, b]x[c, d]$ simultaneously?

Vectorized Newton for nonlinear systems:all zeros over

$[a, b]x[c, d]$

- Can we find all zeros(say real, for example) of a nonlinear system in $[a, b]x[c, d]$ simultaneously?
- $$\begin{aligned} f(x, y) &= 0 \\ g(x, y) &= 0 \end{aligned}$$

Vectorized Newton for nonlinear systems:all zeros over $[a, b]x[c, d]$

- Can we find all zeros(say real, for example) of a nonlinear system in $[a, b]x[c, d]$ simultaneously?
- $$\begin{aligned} f(x, y) &= 0 \\ g(x, y) &= 0 \end{aligned}$$
- it will allow us to determine all stationary points of the system

Vectorized Newton for nonlinear systems:all zeros over

$[a, b]x[c, d]$

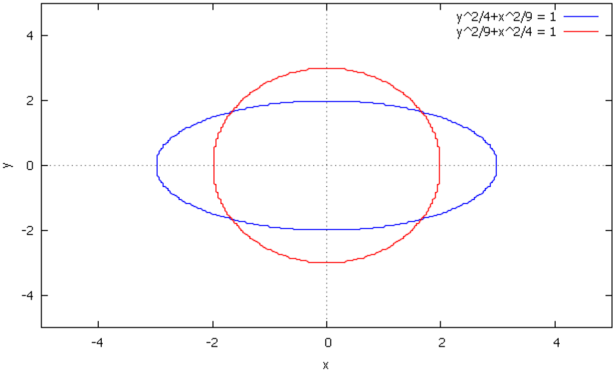
- Can we find all zeros(say real, for example) of a nonlinear system in $[a, b]x[c, d]$ simultaneously?
- $$\begin{aligned} f(x, y) &= 0 \\ g(x, y) &= 0 \end{aligned}$$
- it will allow us to determine all stationary points of the system
- $$\begin{aligned} dx/dt &= f(x, y) \\ dy/dt &= g(x, y) \end{aligned}$$

Vectorized Newton for nonlinear systems

- Can we find all zeros(say real, for example) of a nonlinear system simultaneously?

Vectorized Newton for nonlinear systems

- Can we find all zeros(say real, for example) of a nonlinear system simultaneously?



- .

- Algorithm

Vectorized Newton for nonlinear systems

- Algorithm
  1. input f,g,df,dg,r,dx

Vectorized Newton for nonlinear systems

- Algorithm
  1. input f,g,df,dg,r,dx
  2. set $Z^{(0)}=[(x_0, y_0)^{(0)}, (x_1, y_0)^{(0)}, ..., (x_n, y_0)^{(0)}, ..., (x_0, y_n)^{(0)}, (x_1, y_n)^{(0)}, ..., (x_n, y_n)^{(0)}];$

Vectorized Newton for nonlinear systems

- Algorithm
  1. input f,g,df,dg,r,dx
  2. set $Z^{(0)} = [(x_0, y_0)^{(0)}, (x_1, y_0)^{(0)}, ..., (x_n, y_0)^{(0)}, ..., (x_0, y_n)^{(0)}, (x_1, y_n)^{(0)}, ..., (x_n, y_n)^{(0)}]$;
  3. for $i = 0$ until convergence do

Vectorized Newton for nonlinear systems

- Algorithm
  1. input f,g,df,dg,r,dx
  2. set $Z^{(0)}=[(x_0,y_0)^{(0)},(x_1,y_0)^{(0)},...,(x_n,y_0)^{(0)},...,$
     $(x_0,y_n)^{(0)},(x_1,y_n)^{(0)},...,(x_n,y_n)^{(0)}];$
  3. for $i=0$ until convergence do
     - set $F=\begin{bmatrix} f \\ g \end{bmatrix}$, compute $F(Z^{(i)})$;

Vectorized Newton for nonlinear systems

- Algorithm
  1. input f,g,df,dg,r,dx
  2. set $Z^{(0)}=[(x_0,y_0)^{(0)},(x_1,y_0)^{(0)},\dots,(x_n,y_0)^{(0)},\dots,$
     $(x_0,y_n)^{(0)},(x_1,y_n)^{(0)},\dots,(x_n,y_n)^{(0)}];$
  3. for $i=0$ until convergence do
     - set $F=\begin{bmatrix} f \\ g \end{bmatrix}$, compute $F(Z^{(i)})$;
     - Form the block diagonal Jacobian

$$J(Z^{(i)})=\begin{bmatrix} j(x_0,y_0)^{(i)} & 0 & \cdots & 0 \\ 0 & j(x_1,y_0)^{(i)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & & j(x_n,y_n)^{(i)} \end{bmatrix}$$

with nonsingular j's, where each j is of size 2×2.

Vectorized Newton for nonlinear systems

- Algorithm
  1. input f,g,df,dg,r,dx
  2. set $Z^{(0)}=[(x_0, y_0)^{(0)}, (x_1, y_0)^{(0)}, ..., (x_n, y_0)^{(0)}, ..., (x_0, y_n)^{(0)}, (x_1, y_n)^{(0)}, ..., (x_n, y_n)^{(0)}]$;
  3. for $i = 0$ until convergence do
     - set $F = \begin{bmatrix} f \\ g \end{bmatrix}$, compute $F(Z^{(i)})$;
     - Form the block diagonal Jacobian

$$J(Z^{(i)}) = \begin{bmatrix} j(x_0, y_0)^{(i)} & 0 & \cdots & 0 \\ 0 & j(x_1, y_0)^{(i)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & & j(x_n, y_n)^{(i)} \end{bmatrix}$$

     with nonsingular j's, where each j is of size 2×2.
     - Solve $J(Z^{(i)})\Delta Z = -F(Z^{(i)})$

Vectorized Newton for nonlinear systems

- Algorithm
  1. input f,g,df,dg,r,dx
  2. set $Z^{(0)}=[(x_0,y_0)^{(0)},(x_1,y_0)^{(0)},\ldots,(x_n,y_0)^{(0)},\ldots,$
     $(x_0,y_n)^{(0)},(x_1,y_n)^{(0)},\ldots,(x_n,y_n)^{(0)}];$
  3. for $i=0$ until convergence do
     - set $F=\begin{bmatrix} f \\ g \end{bmatrix}$, compute $F(Z^{(i)})$;
     - Form the block diagonal Jacobian

$$J(Z^{(i)})=\begin{bmatrix} j(x_0,y_0)^{(i)} & 0 & \cdots & 0 \\ 0 & j(x_1,y_0)^{(i)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & & j(x_n,y_n)^{(i)} \end{bmatrix}$$

       with nonsingular j's, where each j is of size 2×2.
     - Solve $J(Z^{(i)})\Delta Z=-F(Z^{(i)})$
     - $Z^{(i+1)}=Z^{(i)}+\Delta Z$

Vectorized Newton for nonlinear systems

- Algorithm
  1. input f,g,df,dg,r,dx
  2. set $Z^{(0)}=[(x_0,y_0)^{(0)},(x_1,y_0)^{(0)},\dots,(x_n,y_0)^{(0)},\dots,$
     $(x_0,y_n)^{(0)},(x_1,y_n)^{(0)},\dots,(x_n,y_n)^{(0)}];$
  3. for $i=0$ until convergence do
     - set $F=\begin{bmatrix} f \\ g \end{bmatrix}$, compute $F(Z^{(i)})$;
     - Form the block diagonal Jacobian

$$J(Z^{(i)})=\begin{bmatrix} j(x_0,y_0)^{(i)} & 0 & \cdots & 0 \\ 0 & j(x_1,y_0)^{(i)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & & j(x_n,y_n)^{(i)} \end{bmatrix}$$

       with nonsingular j's, where each j is of size 2×2.
     - Solve $J(Z^{(i)})\Delta Z=-F(Z^{(i)})$
     - $Z^{(i+1)}=Z^{(i)}+\Delta Z$
     - Accumulate converged components and continue with the ones yet to converge

Vectorized Newton for nonlinear systems

- Algorithm
  1. input f,g,df,dg,r,dx
  2. set $Z^{(0)}=[(x_0,y_0)^{(0)},(x_1,y_0)^{(0)},\dots,(x_n,y_0)^{(0)},\dots,$
     $(x_0,y_n)^{(0)},(x_1,y_n)^{(0)},\dots,(x_n,y_n)^{(0)}]$;
  3. for $i=0$ until convergence do
     - set $F=\begin{bmatrix} f \\ g \end{bmatrix}$, compute $F(Z^{(i)})$;
     - Form the block diagonal Jacobian

$$J(Z^{(i)})=\begin{bmatrix} j(x_0,y_0)^{(i)} & 0 & \cdots & 0 \\ 0 & j(x_1,y_0)^{(i)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & & j(x_n,y_n)^{(i)} \end{bmatrix}$$

       with nonsingular j's, where each j is of size 2×2.
     - Solve $J(Z^{(i)})\Delta Z=-F(Z^{(i)})$
     - $Z^{(i+1)}=Z^{(i)}+\Delta Z$
     - Accumulate converged components and continue with the ones yet to converge
  4. return nonrepeating elements of $Z^{(i+1)}$ in matrix form

Vectorized Newton for nonlinear systems

- Examples: Determine stationary points of the autonomous system

Vectorized Newton for nonlinear systems

- Examples: Determine stationary points of the autonomous system
- $dx/dt = x^2 + y^2 - 1, dy/dt = -x^2 + y$

Vectorized Newton for nonlinear systems

- Examples: Determine stationary points of the autonomous system
- $dx/dt = x^2 + y^2 - 1, dy/dt = -x^2 + y$

W =

-1	1
-1	2
0	1
0	2
1	1
1	2

$i = 0$

W =

-1.0000	1.0000
-0.8333	0.6667
0.8333	0.6667
1.0000	1.0000

$i = 1$

Vectorized Newton for nonlinear systems

- Examples: Determine stationary points of the autonomous system

Vectorized Newton for nonlinear systems

- Examples: Determine stationary points of the autonomous system
- $dx/dt = x^2 + y^2 - 1, dy/dt = -x^2 + y$

Vectorized Newton for nonlinear systems

- Examples: Determine stationary points of the autonomous system
- $dx/dt = x^2 + y^2 - 1, dy/dt = -x^2 + y$

```
W =

   -0.8333    0.6667
   -0.7881    0.6190
    0.7881    0.6190
    0.8333    0.6667
```

$i = 3$

```
W =

   -0.7862    0.6180
    0.7862    0.6180
```

$i = 5$

Vectorized Newton for nonlinear systems

- Examples: Determine stationary points of the autonomous system
- $dx/dt = x^2 + y^2 - 1, dy/dt = -x^2 + y$

```
W =

   -0.8333    0.6667
   -0.7881    0.6190
    0.7881    0.6190
    0.8333    0.6667
```

$i = 3$

```
W =

   -0.7862    0.6180
    0.7862    0.6180
```

$i = 5$

- Needs to be optimized

Vectorized Newton for nonlinear systems

- Examples: Determine stationary points of the autonomous system

Vectorized Newton for nonlinear systems

- Examples: Determine stationary points of the autonomous system
- $dx/dt = x^2 + y^2 - 1, dy/dt = -x^2 - 1 + y$

- Examples: Determine stationary points of the autonomous system
- $dx/dt = x^2 + y^2 - 1, dy/dt = -x^2 - 1 + y$

```
W =

    -1     0
    -1     1
     1     0
     1     1
```

```
W =

   -0.6000    1.2000
   -0.2500    1.0000
    0.2500    1.0000
    0.6000    1.2000
```

-

```
W =

   -0.0001    1.0000
         0    1.0000
    0.0001    1.0000
```

```
ans =

     0     1
```

$i = 14$

Vectorized Newton for nonlinear systems

- Examples: Determine stationary points of the autonomous system

Vectorized Newton for nonlinear systems

- Examples: Determine stationary points of the autonomous system
- $dx/dt = x^2/9 + y^2/4 - 1, dy/dt = x^2/4 + y^2/9 - 1.$

Vectorized Newton for nonlinear systems

- Examples: Determine stationary points of the autonomous system
- $dx/dt = x^2/9 + y^2/4 - 1, dy/dt = x^2/4 + y^2/9 - 1.$

```
>> vnewtons(3)

W =

    -3    -3
    -3     3
     3    -3
     3     3
```

```
W =

   -1.9615   -1.9615
   -1.9615    1.9615
    1.9615   -1.9615
    1.9615    1.9615
```

- $i = 0$ $\qquad$ $i = 1$

```
W =

   -1.6867   -1.6867
   -1.6867    1.6867
    1.6867   -1.6867
    1.6867    1.6867
```

```
ans =

   -1.6641   -1.6641
   -1.6641    1.6641
    1.6641   -1.6641
    1.6641    1.6641
```

$i = 2$ $\qquad$ $i = 5$

Further work

- Optimize the proposed vectorized Newton for nonlinear systems

Further work

- Optimize the proposed vectorized Newton for nonlinear systems
- Solve $y\prime = F(t, y)$ implicitly over a domain with arbitrary set initial values, using vectorized Newton just proposed

- Atkinson,K., An introduction to Numerical Analysis, John Wiley & Sons, 1989.
- Davis, P., Rabinowitz, P, Methods of Numerical Integration, Academic Press, 1984.
- Memoglu, M. Some Vector based zero and extremum finders, M.S. Thesis, KTU, 2012.
- Duchateau,P. & Zachmann, D., Applied PDE, Dover Pub., 1989.
- Coskun, E., Numerical Analysis with Vector Algorithms(textbook under preparation).

Thanks

For your attention My students and family