Laboratorium 3

Artem Buhera
GĆ01 135678 29.03.2021

W tym programie zostały użyte poniższe metody rozwiązywania pojedynczych równań nieliniowych:

- Picarda
  - polega na użyciu metody iteracyjnej $x_{n+1}=\Phi(x_n)$, otrzymywanej poprzez przekształcenie funkcji f(x) do postaci $x=\Phi(x)$
  - metoda iteracyjna $\Phi(x)$ jest zbieżna tylko gdy $|\Phi'(x)| < 1$

- Newtona
  - polega na użyciu metody iteracyjnej $x_{n+1}=\Phi(x_n)=x_n-\frac{f(x_n)}{f'(x_n)}$, gdzie $f'(x_n)\neq 0$
  - metoda Newtona może być rozbieżna gdy $f'(x)\approx 0$ lub niezbieżna

- siecznych
  - polega na użyciu dwuargumentowej metody iteracyjnej
    $$x_{n+2}=\Phi(x_n, x_{n+1})=x_{n+1}-\frac{f(x_{n+1})}{\frac{f(x_{n+1})-f(x_n)}{x_{n+1}-x_n}}$$
  - podobnie do metody Newtona, $\frac{f(x_{n+1})-f(x_n)}{x_{n+1}-x_n}\approx f'(x_{n+1})$

- bisekcji
  - polega na dzieleniu pewnego przedziału [a, b] na dwie równe połowy [a_n, x_n] oraz [x_n, b_n], i wybieraniu jako przedział kolejnej iteracji [a_{n+1}, b_{n+1}] tej połowy, która ma różne znaki wartości funkcji na krańcach
  - zatem f(x) musi być ciągła w przdziale [a, b]

Iterację kończymy wedłuch trzech kryteriów:

- arbitralne ograniczenie na liczbę iteracji: $n \leq n_{max}$
- kryterium dokładności wyznaczenia x_n: $|e_n| \leq TOLX$
  - dla metody bisekcji: $e_n = \frac{b_n-a_n}{2}$
  - dla pozostałych metod: $e_n = x_n - x_{n-1}$
- kryterium wiarygodności x_n jako przybliżenia pierwiastka $|f(x_n)| \leq TOLF$

Szukamy miejsc zerowych dla dwóch funkcji

- $f_1(x)=\sin^2(\frac{x}{4})-x$
- $f_2(x)=\tan(2x)-x-1$

Metoda Picarda

$$f_1:\ \sin^2\left(\frac{x}{4}\right)-x=0 \ \Leftrightarrow\ x=\sin^2\left(\frac{x}{4}\right) \ \Rightarrow\ \Phi(x)=\sin^2\left(\frac{x}{4}\right)$$

$$f_2:\ \tan(2x)-x-1=0 \ \Leftrightarrow\ \tan(2x)=x+1 \ \Leftrightarrow\ \arctan(\tan(2x))=\arctan(x+1) \ \Leftrightarrow$$

$$\Leftrightarrow\ 2x=\arctan(x+1) \ \Leftrightarrow\ x=\frac{1}{2}\arctan(x+1) \ \Rightarrow\ \Phi(x)=\frac{1}{2}\arctan(x+1)$$

```
const double NMAX = 30;
const double TOLX = DBL_EPSILON;
const double TOLF = DBL_EPSILON;


double f1(double x) {
    return sin(x/4.0) * sin(x/4.0) - x;
}

double phi1_Picard(double xn) {
    return sin(xn / 4.0) * sin(xn / 4.0);
}


double f2(double x) {
    return tan(2.0*x) - x - 1.0;
}

double phi2_Picard(double xn) {
    return atan(xn + 1.0) / 2.0;
}
```

```cpp
// pomocnicze funkcje do printowania
void printTOLX(int n, double estimator) {
    cout << "przekroczono TOLX po " << n
         << " iteracjach - estymator = " << estimator << endl;
}

void printTOLF(int n, double residuum) {
    cout << "przekroczono TOLF po " << n
         << " iteracjach - residuum = " << residuum << endl;
}

void printTOLXAndTOLF(int n, double estimator, double residuum) {
    cout << "przekroczono TOLX oraz TOLF po " << n
         << " iteracjach - estymator = " << estimator
         << ", residuum = " << residuum << endl;
}

void printNMAX() {
    cout << "przekroczono maksymalną liczbę iteracji - " << NMAX << endl;
}


void printHeader() {
    cout << endl << "n   x_n                     x_n1                    "
                    "en                        residuum" << endl;
}

void printIteration(int n, double xn, double xn1, double en, double residuum) {
    cout << setw(2) << left << n << "  "
         << setprecision(17) << setw(24) << left << xn
         << setprecision(17) << setw(24) << left << xn1
         << setprecision(17) << setw(24) << left << en  << "  "
         << setprecision(17) << setw(24) << left << residuum << endl;
}
```

```cpp
double solvePhi(double (*phi)(double), double (*f)(double), double x0) {
    double xn = x0, xn1, en, residuum;
    cout << endl;

    // arbitralne ograniczenie na liczbę iteracji
    for (int n = 1; n <= NMAX; n++) {
        xn1 = phi(xn);

        en = xn1 - xn;
        residuum = f(xn1);

        cout << setprecision(17) << "Phi(" << xn << ") = "
             << setprecision(17) << xn1 << endl;

        // kryterium dokładności wyznaczenia xn1
        // oraz wiarygodności xn1 jako przybliżenia pierwiastków
        if (abs(en) <= TOLX && abs(residuum) <= TOLF) {
            printTOLXAndTOLF(n, en, residuum);
            return xn1;

        } else {
            // kryterium dokładności wyznaczenia xn
            if (abs(en) <= TOLX) {
                printTOLX(n, en);
                return xn1;
            }

            // kryterium wiarygodności xn jako przybliżenia pierwiastka
            if (abs(residuum) <= TOLF) {
                printTOLF(n, residuum);
                return xn1;
            }
        }

        xn = xn1;
    }

    printNMAX();
    return xn1;

}

int main() {
    cout << endl << "Metoda Picarda dla f1: ";
    double root1_Picard = solvePhi(phi1_Picard, f1, 1.0);
    cout << "x0 = " << root1_Picard << ", f1(x0) = " << f1(root1_Picard) << endl;

    cout << endl << "Metoda Picarda dla f2: ";
    double root2_Picard = solvePhi(phi2_Picard, f1, 1.0);
    cout << "x0 = " << root2_Picard << ", f2(x0) = " << f2(root2_Picard) << endl;
};
```

Wynik działania dla początkowego punktu $x_0 = 1$:

```
Metoda Picarda dla f1:
n   x_n1                    en                        residuum
1   0.061208719054813648    -0.93879128094518638      -0.060974580625176605
2   0.00023413842963704515  -0.060974580625176605     -0.00023413500333367845
3   3.4263002606431685e-09  -0.00023413500333367845   -3.4263002599094477e-09
4   7.3372084225521526e-19  -3.4263002599094477e-09   -7.3372084225521526e-19
przekroczono TOLF po 4 iteracjach - residuum = -7.3372084225521526e-19
x0 = 7.3372084225521526e-19, f1(x0) = -7.3372084225521526e-19

Metoda Picarda dla f2:
n   x_n1                    en                        residuum
1   0.5535743588970452      -0.4464256411029548       0.44642564110295435
2   0.49943949268161769     -0.054134866215427513     0.054134866215427291
3   0.49131060741579274     -0.0081288852658249477    0.0081288852658247812
4   0.49005465229082668     -0.0012559551249660683    0.0012559551249660128
5   0.48985975739491727     -0.00019489489590940323   0.00019489489590918119
6   0.48982949395354863     -3.0263441368638677e-05   3.0263441368694188e-05
7   0.48982479413159841     -4.6998219502270011e-06   4.6998219502825123e-06
8   0.48982406425149455     -7.298801038557734e-07    7.2988010368923995e-07
9   0.48982395090117831     -1.1335031624426506e-07   1.1335031624426506e-07
10  0.48982393329787727     -1.7603301039059716e-08   1.760330126110432e-08
11  0.48982393056408491     -2.7337923591552737e-09   2.7337923036441225e-09
12  0.48982393013952696     -4.2455794435625194e-10   4.2455772231164701e-10
13  0.48982393007359315     -6.5933813964136334e-11   6.5933924986438797e-11
14  0.48982393006335362     -1.0239531444966588e-11   1.0239586956117819e-11
15  0.48982393006176339     -1.590227949321843e-12    1.5902834604730742e-12
16  0.48982393006151648     -2.4691360067663481e-13   2.4691360067663481e-13
17  0.48982393006147812     -3.8358205500799158e-14   3.8413716652030416e-14
18  0.48982393006147218     -5.9396931817445875e-15   6.2172489379008766e-15
19  0.48982393006147124     -9.4368957093138306e-16   8.8817841970012523e-16
20  0.48982393006147107     -1.6653345369377348e-16   0
przekroczono TOLX oraz TOLF po 20 iteracjach - estymator = -1.6653345369377348e-16,
residuum = 0
x0 = 0.48982393006147107, f2(x0) = 0
```

W obu przypadkach otrzymaliśmy dobre wyniki. Szczególnie w przypadku funkcji f_1 widzimy, że po każdej iteracji dostajemy coraz dokładniejsze miejsce zerowe.

Ale gdybyśmy dla funkcji f_2 wyznaczyli $\Phi(x)$ w inny sposób taki, że $|\Phi'(x)| > 1$, to możemy zaobserwować rozbieżność, a więc nieefektywność metody:

$$f_2:\ \tan(2x)-x-1=0 \ \Leftrightarrow\ x=\tan(2x)-1 \ \Rightarrow\ \Phi(x)=\tan(2x)-1$$

$$\Phi'(x)=(\tan(2x)-1)'=(\tan(2x))'-(1)'=\frac{1}{\cos^2(2x)}(2x)'-0=\frac{2}{\cos^2(2x)}$$

$$\forall x\in\mathbb{R}\ \ \Phi'(x)\geq 2$$

```
double phi2_Picard_divergent(double xn) {
    return 2.0 / (cos(2*xn) * cos(2*xn));
}

int main() {
    cout << endl << "Metoda Picarda dla f2: ";
    double root2_Picard = solvePhi(phi2_Picard_divergent, f1, 1.0);
    cout << "x0 = " << root2_Picard << ", f2(x0) = " << f2(root2_Picard) << endl;
};
```

Wynik działania dla początkowego punktu $x_0 = 1$:

```
Metoda Picarda dla f2:
n   x_n1                    en                      residuum
1   11.548798408083835      10.548798408083835      -11.485466638740903
2   9.9720445885267814      -1.5767538195570534     -9.6071602638284546
3   9.5153461252384623      -0.45669846328831909    -9.0379804286726912
4   2.0670832505329004      -7.4482628747055619     -1.8229729459554163
5   6.6956395025627637      4.6285562520298633      -5.7062342772274786
6   4.3428141178645339      -2.3528253846982299     -3.5602409043426388
7   3.6617795903359682      -0.68103452752856564    -3.0331941515019301
8   7.8145581675655986      4.1527785772296308      -6.9541047629578108
9   2.0124853918737142      -5.8020727756918848     -1.7800047740713396
10  4.9671435011325373      2.9546581092588231      -4.0715414504909804
11  2.6243233869698903      -2.342820114162647      -2.2522038049640241
12  7.6616029039805991      5.0372795170107088      -6.7757282025899519
13  2.3279522977802585      -5.3336506062003401     -2.0257978160909067
14  627.53088980263578      625.20293750485553      -627.49261471249429
15  99405.506059526757      98777.975169724115      -99404.54253697941
16  1790.7428267602411      -97614.76323276652      -1789.7429033829831
17  2.0098401161942467      -1788.7329866440468     -1.7779179646496781
18  4.903877183750275       2.8940370675560283      -4.0181440980376184
19  2.3246053699060472      -2.5792718138442279     -2.0232190518202251
20  501.73320688114615      499.40860151124008      -501.68105344041101
21  27.622491342492363      -474.11071553865378     -27.282582188655443
22  28.707743912608731      1.0852525701163671      -28.100237474139373
23  4.7739819703627466      -23.933761942245983     -3.9097097416003304
24  2.0306592646904393      -2.7433227056723073     -1.7943292080014659
25  5.4454021306565892      3.4147428659661498      -4.4886319577602674
26  182.87128667039448      177.42588453973789      -181.89817749057016
27  31.928105846987542      -150.94318082340695     -30.944411916658677
28  7.4065027616016321      -24.521603085385909     -6.4833163351724146
29  5.110971532767862       -2.2955312288337701     -4.1944213563434678
30  4.096395185388948       -1.014576347378914      -3.3666589079439331
przekroczono maksymalną liczbę iteracji - 30
x0 = 4.096395185388948, f2(x0) = -7.9341028561235145
```

Obliczone rozwiązanie ewidentnie nie jest miejscem zerowym funkcji f_2

Metoda Newtona

```
double phi1_Newton(double xn) {
    return xn - f1(xn) / f1_derivative(xn);
}

double phi2_Newton(double xn) {
    return xn - f2(xn) / f2_derivative(xn);
}

int main() {
    cout << endl << "Metoda Newtona dla f1: ";
    double root1_Newton = solvePhi(phi1_Newton, f1, -1.0);
    cout << "x0 = " << root1_Newton << ", f1(x0) = " << f1(root1_Newton) << endl;

    cout << endl << "Metoda Newtona dla f2: ";
    double root2_Newton = solvePhi(phi2_Newton, f2, -1.0);
    cout << "x0 = " << root2_Newton << ", f2(x0) = " << f2(root2_Newton) << endl;
};
```

Wynik działania programu dla początkowego punktu x_0 = -1:

```
Metoda Newtona dla f1:
n   x_n1                    en                        residuum
1   -0.052370702529424662   0.94762929747057534       0.052542110890134636
2   -0.0001702742269177257  0.052200428302506936      0.00017027603899974662
3   -1.8120434506997598e-09 0.000170272414874275      1.8120434509049785e-09
4   -2.0521896048103712e-19 1.8120434504945408e-09    2.0521896048103712e-19
przekroczono TOLF po 4 iteracjach - residuum = 2.0521896048103712e-19
x0 = -2.0521896048103712e-19, f1(x0) = 2.0521896048103712e-19

Metoda Newtona dla f2:
n   x_n1                    en                        residuum
1   -1.8148751603810271     -0.31487516038102714      0.28385117509569735
2   -1.9255827150570513     -0.11070755467602411      0.066796628907151101
3   -1.9448046167290551     -0.019221901672003883     0.016906103620519142
4   -1.9493468368075397     -0.0045422200784845934    0.0043980266080048125
5   -1.9505085360422967     -0.0011616992347569521    0.0011520142051775206
6   -1.9508114939272023     -0.00030295788490564135   0.00030229468912068569
7   -1.9508909002347303     -7.9406307528007147e-05   7.9360665851124779e-05
8   -1.9509117402796241     -2.0840044893777332e-05   2.0836899667031616e-05
9   -1.9509172115984674     -5.4713188433197502e-06   5.4711020274211819e-06
10  -1.950918648161543      -1.4365630756074665e-06   1.4365481280087522e-06
11  -1.9509190253580768     -3.7719653378509577e-07   3.7719550327608431e-07
12  -1.9509191243987072     -9.9040630363589344e-08   9.9040559753404978e-08
13  -1.950919150403884l     -2.6005176900767424e-08   2.6005171793741511e-08
14  -1.9509191572320868     -6.8282026699506559e-09   6.8282028919952609e-09
15  -1.9509191590249744     -1.7928876161477092e-09   1.7928876161477092e-09
16  -1.9509191594957345     -4.7076009757063275e-10   4.7076076370444753e-10
17  -1.9509191596193425     -1.2360801271427135e-10   1.2360823475887628e-10
18  -1.9509191596517985     -3.2456037857286901e-11   3.2455593768077051e-11
19  -1.9509191596603204     -8.5218498924177766e-12   8.5220719370227016e-12
20  -1.9509191596625579     -2.2375434838295405e-12   2.2377655284344655e-12
21  -1.9509191596631454     -5.8753002463163284e-13   5.879741138414829e-13
22  -1.9509191596632998     -1.5432100042289676e-13   1.5454304502782179e-13
23  -1.9509191596633404     -4.063416270128072 9e-14   4.0412118096355698e-14
24  -1.9509191596633511     -1.0658141036401503e-14   1.021405182655144e-14
25  -1.9509191596633537     -2.6645352591003757e-15   2.886579864025407e-15
```

```
26  -1.9509191596633544     -6.661338147750939 2e-16   8.881784197001252 3e-16
27  -1.9509191596633546     -2.220446049250313 1e-16   4.440892098500626 2e-16
przekroczono TOLX po 27 iteracjach - estymator = -2.220446049250313 1e-16
x0 = -1.9509191596633546, f2(x0) = 4.440892098500626 2e-16
```

```
przekroczono TOLX po 29 iteracjach - estymator = -2.2204460492503131e-16
x0 = -1.9509191596633546, f2(x0) = 4.4408920985006262e-16
```

Widzimy, że dla pierwszej funkcji rozwiązanie jest obliczone bardzo szybko, ale dla drugiej - prawie przekracza ustalony limit iteracji. Spowodowane jest to tym, że estymator błędu i residuum zmniejszają się dużo wolniej.

Metoda siecznych

```c
double solveSecant(double (*f)(double), double x0, double x1) {
    double xn = x0, xn1 = x1, xn2, en, residuum;

    printHeaderSecant();

    // arbitralne ograniczenie na liczbę iteracji
    for (int n = 0; n < NMAX; n++) {
        xn2 = xn1 - f(xn1) / (((f(xn1) - f(xn)) / (xn1 - xn)));

        en = xn2 - xn1;
        residuum = f(xn2);

        printIteration(n, xn2, en, residuum);

        // kryterium dokładności wyznaczenia xn1
        // oraz wiarygodności xn1 jako przybliżenia pierwiastków
        if (abs(en) <= TOLX && abs(residuum) <= TOLF) {
            printTOLXAndTOLF(n, en, residuum);
            return xn2;

        } else {
            // kryterium dokładności wyznaczenia xn
            if (abs(en) <= TOLX) {
                printTOLX(n, en);
                return xn2;
            }

            // kryterium wiarygodności xn jako przybliżenia pierwiastka
            if (abs(residuum) <= TOLF) {
                printTOLF(n, residuum);
                return xn2;
            }
        }

        xn = xn1;
        xn1 = xn2;
    }

    printNMAX();
    return xn2;
}
```

Wynik działania programu dla punktów początkowych $x_0 = 2$, $x_1 = 1$:

```
Metoda siecznych dla f1:
n   x_n2                    en                          residuum
0   -0.12922371234889174    -1.1292237123488917         0.13026702230455392
1   0.0083745926979195695   0.13759830504681131         -0.0083702093416456857
2   6.7108448402642892e-05  -0.0083074842495169266      -6.7108166931152476e-05
3   -3.5143860481519763e-08 -6.7143592263124412e-05     3.5143860558712946e-08
4   1.4740373965443562e-13  3.5144007885259418e-08      -1.4740373965443426e-13
5   3.2377103274838187e-22  -1.4740373933066459e-13     -3.2377103274838187e-22
przekroczono TOLF po 5 iteracjach - residuum = -3.2377103274838187e-22
x0 = 3.2377103274838187e-22, f1(x0) = -3.2377103274838187e-22

Metoda siecznych dla f2:
n   x_n2                    en                          residuum
0   2.7862944507408782      1.7862944507408782          -4.6468606719417034
1   -15.187476491667617     -17.973770942408496         15.894896513462722
2   -1.2796483293690599     13.907828162298557          0.93810326841295288
3   -0.40733707642744899    0.87231125294161094         -1.65299852546827
4   -0.963829769330104      -0.55649269290265502        2.6460471615540353
5   -0.62131051792374148    0.34251925140636252         -3.3156561666228619
6   -0.81180575151662882    -0.19049523359288734        18.728149195186123
7   -0.6499633156420882     0.16184243587454061         -3.9511140650051719
8   -0.67815902541282969    -0.028195709770741484       -4.9126048682145171
9   -0.53409692869784997    0.14406209671497971         -2.2851219247194781
10  -0.40880612270442962    0.12529080599342035         -1.6577904263982304
11  -0.077711798176750613   0.33109432452767901         -1.0789755066965787
12  0.5394849160263897      0.61719671420314026         0.32710408816341507
13  0.39590302410868755     -0.14358189191770215        -0.38300442518906408
14  0.47334541347235515     0.077442389363667596        -0.084697984844645857
15  0.49533358883016487     0.021988175357809725        0.030561498792084141
16  0.48950333891549591     -0.0058302499146689613      -0.0017417737624689789
17  0.489817702616971       0.00031436370147508441      -3.3871264966123249e-05
18  0.48982393710415278     6.2344871817887615e-06      3.8306207983396234e-08
19  0.48982393006131636     -7.0428364251640119e-09     -8.4132700806094363e-13
20  0.48982393006147101     1.5465406733028431e-13      -2.2204460492503131e-16
przekroczono TOLF po 20 iteracjach - residuum = -2.2204460492503131e-16
x0 = 0.48982393006147101, f2(x0) = -2.2204460492503131e-16
```

Otrzymaliśmy dobre wyniki. Możemy zauważyć, szczególnie dla drugiej funkcji, jak metoda "kalibruje" kierunek szukania rozwiązania, zanim nie zaczyna liczyć coraz bardziej precyzyjne przybliżenia w ostatnich iteracjach.

Metoda bisekcji

```
double solveBisection(double (*f)(double), double a0, double b0) {
    double xn, an = a0, bn = b0, f_xn, f_an, f_bn, en, residuum;

    // sprawdzenie poprawności podanego przedziału [an, bn]
    double f_a0 = f(a0), f_b0 = f(b0);
    if ((f_a0 < 0.0 && f_b0 < 0.0) || (f_a0 >= 0.0 && f_b0 >= 0.0)) {
        cout << "podany przedział jest zły - f(a0) oraz f(b0) są tego samego znaku"
             << endl;
        return -1.0;
    }

    printHeader();

    // arbitralne ograniczenie na liczbę iteracji
    for (int n = 0; n < NMAX; n++) {
        xn = (an + bn) / 2.0;

        f_xn = f(xn);
        f_an = f(an);
        f_bn = f(bn);

        if ((f_an <= 0.0 && f_xn >= 0.0) || (f_an >= 0.0 && f_xn <= 0.0))
            bn = xn;
        else if ((f_xn <= 0.0 && f_bn >= 0.0) || (f_xn >= 0.0 && f_bn <= 0.0))
            an = xn;

        en = (bn - an) / 2.0;
        residuum = f_xn;

        printIteration(n, xn, en, residuum);

        // kryterium dokładności wyznaczenia xn
        // oraz wiarygodności xn jako przybliżenia pierwiastków
        if (abs(en) <= TOLX && abs(residuum) <= TOLF) {
            printTOLXAndTOLF(n, en, residuum);
            return xn;

        } else {
            // kryterium dokładności wyznaczenia xn
            if (abs(en) <= TOLX) {
                printTOLX(n, en);
                return xn;
            }

            // kryterium wiarygodności xn jako przybliżenia pierwiastka
            if (abs(residuum) <= TOLF) {
                printTOLF(n, residuum);
                return xn;
            }
        }
    }

    printNMAX();
    return xn;
}
```

```cpp
int main() {
    cout << endl << "Metoda bisekcji dla f1: ";
    double root1_Bisection = solveBisection(f1, -1.5, 2.0);
    cout << "x0 = " << root1_Bisection << ", f1(x0) = " << f1(root1_Bisection) << endl;

    cout << endl << "Metoda bisekcji dla f2: ";
    double root2_Bisection = solveBisection(f2, -0.6, 0.7);
    cout << "x0 = " << root2_Bisection << ", f2(x0) = " << f2(root2_Bisection) << endl;
}
```

Wynik działania programu dla przedziałów początkowych [-1.5, 2] dla funkcji f_1 oraz [-0.6, 0.7] dla f_2:

```
Metoda bisekcji dla f1:
n   x_n1                    en                      residuum
0   0.25                    0.875                   -0.24609883361466453
1   -0.625                  0.4375                  0.64921602597591388
2   -0.1875                 0.21875                 0.18969565677099912
3   0.03125                 0.109375                -0.031188966085503329
4   -0.078125               0.0546875               0.078506421222645506
5   -0.0234375              0.02734375              0.023471831882490711
6   0.00390625              0.013671875             -0.0039052963259867588
7   -0.009765625            0.0068359375            0.00977158545263517
8   -0.0029296875           0.00341796875           0.0029302239417070552
9   0.00048828125           0.001708984375          -0.00048826634883888015
10  -0.001220703125         0.0008544921875         0.0012207962572545704
11  -0.0003662109375        0.00042724609375        0.00036621931940314812
12  6.103515625e-05         0.000213623046875       -6.1034923419356366e-05
13  -0.000152587890625      0.0001068115234375      0.00015258934581652213
14  -4.57763671875e-05      5.340576171875e-05      4.5776498154737049e-05
15  7.62939453125e-06       2.6702880859375e-05     -7.6293908932711929e-06
16  -1.9073486328125e-05    1.33514404296875e-05    1.9073509065492544e-05
17  -5.7220458984375e-06    6.67572021484375e-06    5.722047944800579e-06
18  9.5367431640625e-07     3.337860107421875e-06   -9.5367425956283114e-07
19  -2.384185791015625e-06  1.6689300537109375e-06  2.3841861462869929e-06
20  -7.152557373046875e-07  8.3446502685546875e-07  7.1525576927911061e-07
21  1.1920928955078125e-07  4.1723251342773438e-07  -1.1920928866260283e-07
22  -2.9802322387695312e-07 2.0861625671386719e-07  2.9802322942806825e-07
23  -8.9406967163085938e-08 1.0430812835693359e-07  8.9406967662686299e-08
24  1.4901161193847656e-08  5.2154064178466797e-08  -1.4901161179969868e-08
25  -3.7252902984619141e-08 2.6077032089233398e-08  3.7252903071355314e-08
26  -1.1175870895385742e-08 1.3038516044616699e-08  1.1175870903191998e-08
27  1.862645149230957e-09   6.5192580223083496e-09  -1.8626451490141166e-09
28  -4.6566128730773926e-09 3.2596290111541748e-09  4.6566128744326453e-09
29  -1.3969838619232178e-09 1.6298145055770874e-09  1.3969838620451905e-09
przekroczono maksymalną liczbę iteracji - 30
x0 = -1.3969838619232178e-09, f1(x0) = 1.3969838620451905e-09
```

```
Metoda bisekcji dla f2:
n   x_n1                    en                        residuum
0   0.049999999999999989    0.3249999999999996        -0.9496653279145495
1   0.375                   0.1624999999999998        -0.44340354005592753
2   0.5374999999999998      0.08124999999999989       0.3114185102106716 8
3   0.4562499999999999      0.04062499999999994       -0.16322232739157494
4   0.4968749999999996      0.02031249999999983       0.039329345355359591
5   0.4765625               0.01015624999999978       -0.068903487218315629
6   0.4867187499999998      0.005078124999999889      -0.016706492569152798
7   0.4917968749999997      0.002539062499999944      0.010806336459785459
8   0.4892578125            0.001269531249999833      -0.0030730615973013631
9   0.4905273437499996      0.0006347656249999778     0.003835489240867628 9
10  0.4898925781249998      0.0003173828124999889     0.00037347760005967956
11  0.4895751953124999      0.00015869140624999445    -0.0013517197483285948
12  0.4897338867187499 6    7.934570312501110 2e-05   -0.00048960379767071238
13  0.4898132324218749 7    3.967285156250555 1e-05   -5.818387813705250 3e-05
14  0.489852905273437 5     1.983642578126665 3e-05   0.00015761665381108791
15  0.4898330688476562 1    9.918212890619448 9e-06   4.970883758947941 5e-05
16  0.4898231506347655 6    4.959106445323602 2e-06   -4.239407643491510 7e-06
17  0.4898281097412108 8    2.479553222661801 1e-06   2.273424310650362 7e-05
18  0.4898256301879881 9    1.239776611317022 8e-06   9.24729976792271 2e-06
19  0.4898243904113768 8    6.198883056585113 8e-07   2.503916571638953 3e-06
20  0.4898237705230712 4    3.09944152815377 9e-07    -8.677529084177848 5e-07
21  0.4898240804672240 3    1.549720763938111 6e-07   8.180799884183187 5e-07
22  0.4898239254951476 4    7.748603819690558 2e-08   -2.483692063126596 8e-08
23  0.4898240029811858 4    3.874301909845279 1e-08   3.966214185968652 8e-07
24  0.4898239642381667 6    1.937150956310418 3e-08   1.858922202835344 7e-07
25  0.489823944866657 2     9.685754781552091 7e-09   8.05276425541734 4e-08
26  0.4898239351809023 9    4.84287737689825 8e-09    2.784535912958574 5e-08
27  0.4898239303380250 2    2.42143868844912 9e-09    1.504218749559527 2e-09
28  0.4898239279165863 5    1.210719330346776 7e-09   -1.166635099636437 2e-08
29  0.4898239291273056 6    6.053596790511761 6e-10   -5.081066234424724 8e-09
przekroczono maksymalną liczbę iteracji - 30
x0 = 0.48982392912730566, f2(x0) = -5.0810662344247248e-09
```

Poza tym że metoda bisekcji w danych przypadkach jest zbieżna, precyzyjność wyników rośnie zbyt wolnie w stosunku do ilości iteracji.

Zwróćmy też uwagę na to, że musimy wiedzieć, czy w początkowym przedziale [a_0, b_0] mieści się rozwiązanie o nieparzystej krotności, żeby jego się dało znaleźć tą metodą, oraz że funkcja w podanym przedziale jest zbieżna. W innym przypadku metoda nie da poprawnych wyników.

Porównanie metod

```
Dla funkcji f1(x) = sin(x/4)^2 - x:
Picard)    x0 =  7.3372084225521526e-19, f1(x0) = -7.3372084225521526e-19 - TOLF n=4
Newtona)   x0 = -2.0521896048103712e-19, f1(x0) =  2.0521896048103712e-19 - TOLF n=4
Siecznych) x0 =  3.2377103274838187e-22, f1(x0) = -3.2377103274838187e-22 - TOLF n=5
Bisekji)   x0 = -1.3969838619232178e-09, f1(x0) =  1.3969838620451905e-09 - NMAX

Dla funkcji f2(x) = tan(2x) - x - 1:
Picard)    x0 =  0.48982393006147107, f2(x0) =  0              - TOLX oraz TOLF n=20
Newtona)   x0 = -1.9509191596633546,  f2(x0) =  4.4408920985006262e-16  - TOLX n=29
Siecznych) x0 =  0.48982393006147101, f2(x0) = -2.2204460492503131e-16  - TOLF n=20
Bisekji)   x0 =  0.48982392912730566, f2(x0) = -5.0810662344247248e-09  - NMAX
```

Dla obu funkcji metoda bisekcji szukała rozwiązania najwolniej - residuum zmniejszał się co iterację jedynie o dwa razy.

Dla pierwszej funkcji wszystkie pozostałe metody mieli optymalne rozwiązanie, ale szukanie dla drugiej funkcji zajęło dużo więcej iteracji.

Ilość potrzebnych iteracji w zależności od dobrze wybranego miejsca zerowego okazała się największa dla metody Newtona.