PYTHON

ZVEZA ZA TEHNIČNO KULTURO SLOVENIJE

Numerika 1

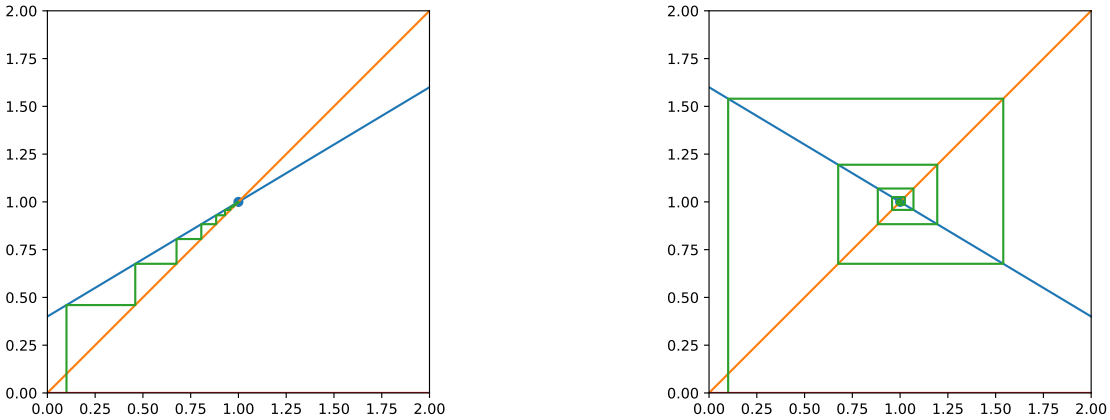
Python urejevalnik se nahaja na strani:
`https://www.w3resource.com/python-exercises/python-basic-exercises.php#EDITOR`

1. **Iteracija** Iščemo presečišče grafa funkcije $y = f(x)$ s premico $y = x$. Presečišče bomo poskušali poiskati iterativno. Rešili bomo enačbo $f(x) = x$, tako da izberemo začetni približek x_0 in ponavljamo postopek:

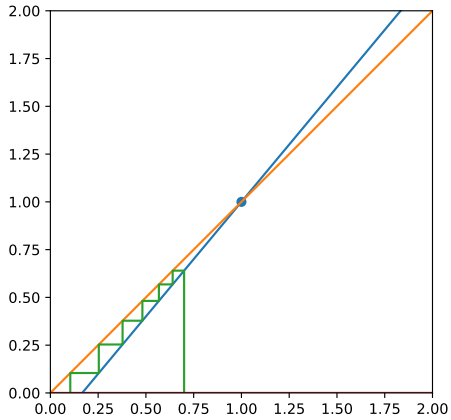
   $$x = x_0, \quad x_n = f(x_{n-1}), \quad n = 1, 2, \cdots$$

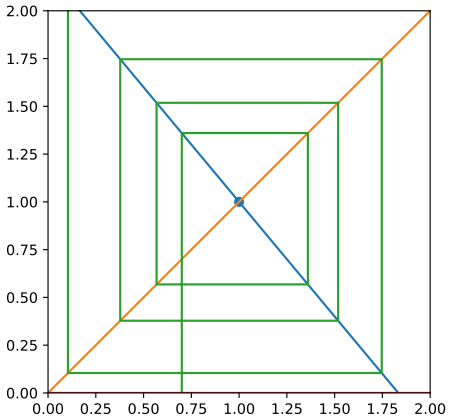
   Opažamo, da se v nekaterih primerih zaporedje x_n v resnici začne približevati rešitvi, v drugih primerih pa se od te oddaljuje. Vedenje zaporedja bomo proučili na najpreprostejšemu primeru, ko je funkcija $f(x)$ linearna. Vzelo bomo:

   $$f(x) = a(x-1) + 1, \quad f(x) = x, \quad a(x-1) + 1 = x, \quad x = 1$$



Koeficient $a = 0.6$ levo in $a = -0.6$ desno



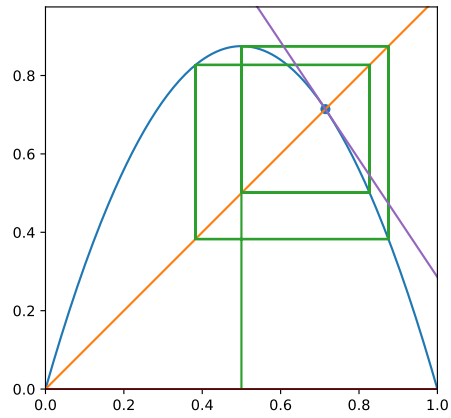


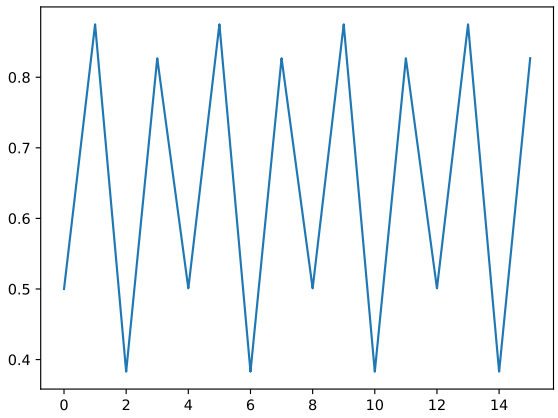
Koeficient $a = 1.2$ levo in $a = -1.2$ desno

2. **Logistična enačba** Reševali bomo iterativno logistično enačbo $f(a, x) = x$, $f(a, x) = ax(x - 1)$. Enačba predstavlja preprost zakon rasti populacije. Zaporedje x_n predstavlja stanje populacije v času n.

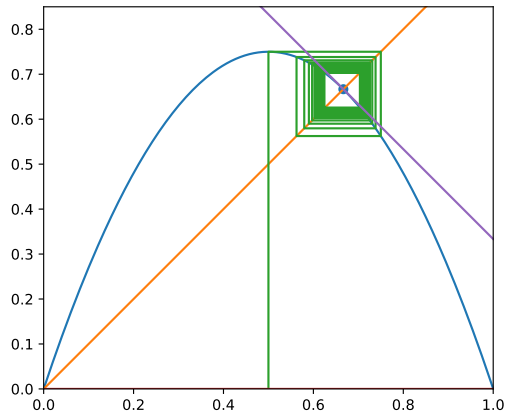
$$x_n = ax_{n-1}\left(1 - x_{n-1}\right)$$

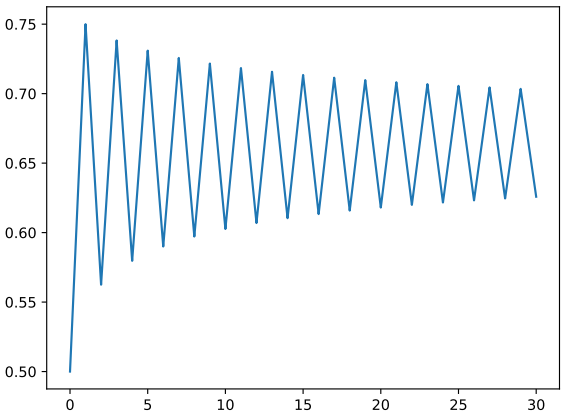
Prvi faktor x trend rasti, medtem ko je drugi faktor $(x-1)$ trend zmanjševanja populacije. Koeficient a določa vedenje populacije. Za nekatere vrednosti populacija narašča, za druge pada, lahko pa je stacionarna, oziroma niha okoli ene vrednosti, ali pa se vede kaotično.



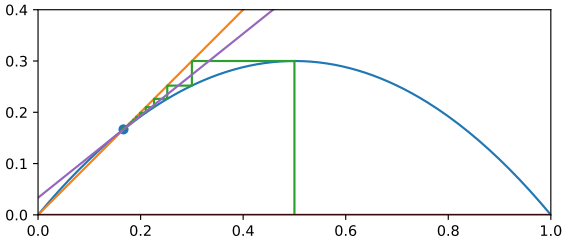


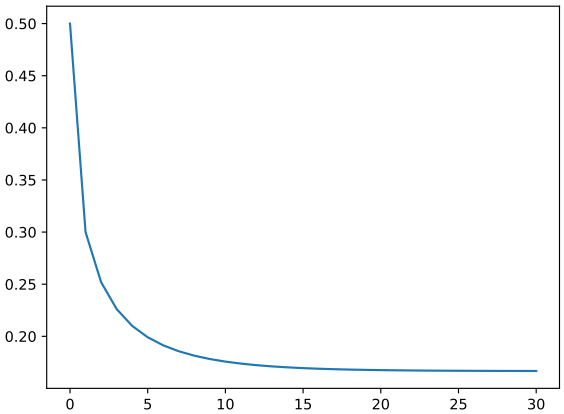
Koeficient $a = 3.5$





Koeficient $a = 3$





Koeficient $a = 1.2$

Poiščemo točko presečišča T premice $y = x$ s parabolo $y = a\,x(1-x)$. Rešimo enačbo

$$x \quad = \quad a\,x(1-x), \to 1 = a(1-x), \to 1-a = -a\,x, \to x = \frac{a-1}{a}$$

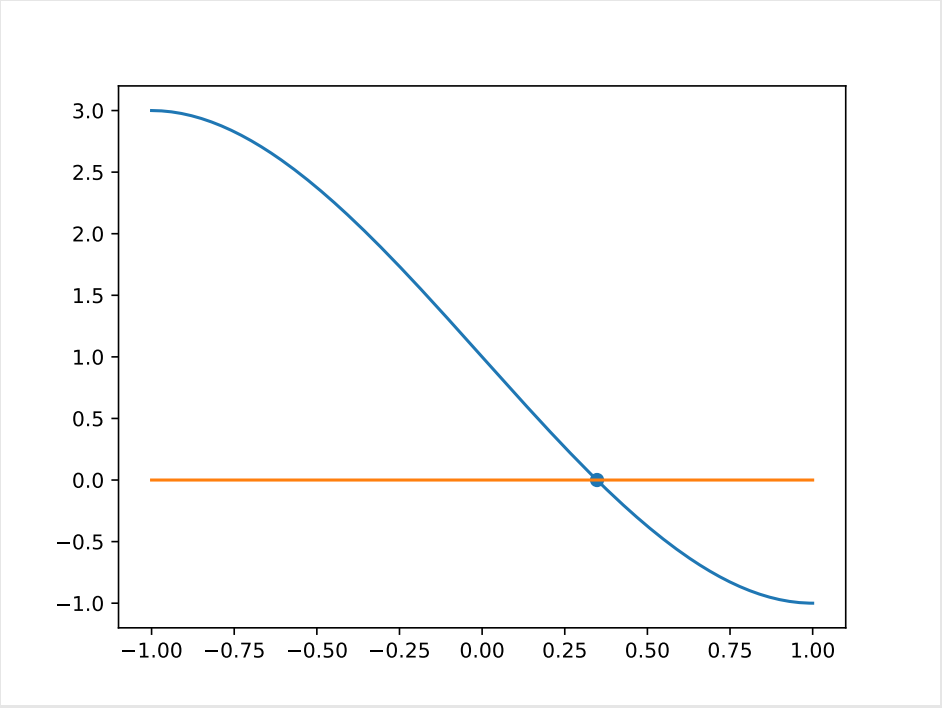
3. **Bisekcija** Dana je funkcija

$$f(x) = x^3 - 3\,x + 1.$$

Narišite graf funkcije $f(x)$ in poišči absciso (koordinato x) točke, kjer graf funkcije seka os x. Uporabi metodo bisekcije. Na začetku podamo interval (a, b) na katerem se nahaja ta točka, ki je

hkrati edina točka na intervalu, kjer graf funkcije seka os x. To pomeni, da sta funkcijski vrednosti $f(a)$ in $f(b)$ različnega znaka. Poišči točko $c = (a + b)/2$. Glede na predznak vrednosti $f(c)$ se odločite, ali postane $a = c$ oziroma $b = c$. Nato nadaljujemo v zanki toliko časa, da postane interval (a, b) dovolj majhen, oziroma da $|f(c)|$ pade pod določeno vrednost.

Graf



4. **Top** Ciljate točko `p`, ki se nahaja nekje med 264 in 480 metri od topa. Funkcija `top` simulira top. Parameter, ki ga sprejema, je količina smodnika. Vemo, da je količina *smodnik* = 10 premajhna, krogla pristane v vsakem primeru pred ciljem, in de je količina *smodnik* = 16 prevelika, krogla pristane za ciljem. Napišite program, ki bo poiskal ravno pravšnjo količino smodnika, da bo krogla zadela cilj. Pri vsakem zagonu programa se položaj točke p izbere naključno. Vzeli bomo, da je cilj zadet, če krogla pade na mesto, ki je oddaljeno za manj ko meter od cilja.

   Izpišite vrstice s količinami smodnika pri posameznih poskusih. Prvi stolpec je položaj točke `p`, drugi stolpec so mesta, kjer je pristala krogla, in v zadnjem stolpcu so količine smodnika.

   Animacija:
   `https://scratch.mit.edu/projects/674822155/editor`

   Procedura `top`
   V osi y deluje gravitacija v nasprotni smeri osi. V tej smeri je gibanje sestavljeno iz dveh delov $y = y_1 + y_2$, to je gibanje v smeri osi y z začetno hitrostjo v_{y0}, $y_1(t) = v_{y0}\,t$ in prostega pada $y_2(t) = \frac{g}{2}t^2$. Gibanje v smeri osi x je enakomerno gibanje z začetno hitrostjo v_{x0}. Težni pospešek je obrnjen navzdol, torej $g < 0$.

$$y(t) = y_0 + v_{y0}\,t + \frac{g}{2}t^2, \quad v(t) = v_{y0} + g\,t,$$
$$x(t) = x_0 + v_{x0}\,t, \quad v_x = v_{x0}$$

Izrazimo t iz druge enačbe in ga vstavimo v prvo enačbo. Rezultat je kvadratna funkcija za poševni met $y = f(x)$. Zaradi preglednosti bomo vzeli, da je $x_0 = y_0 = 0$:

$$y(x) = \frac{v_{y0}}{v_{x0}}x + \frac{g}{2}\left(\frac{x}{v_{x0}}\right)^2.$$

Trajektorijo za poševni met gradimo korakoma v časovnih intervalih dt. Začnemo v točki $(0, 0)$. Začetni hitrosti v smeri oseh sta (v_{x0}, v_{y0}). Časovni interval označimo z dt.

$$\begin{aligned}
&x_0 = 0, \quad x_n = x_{n-1} + v_{x0}dt \\
&v_{y(n-1/2)} = v_{y(n-1)} + g\frac{dt}{2} \\
&y_0 = 0, \quad y_n = y_{n-1} + v_{y(n-1/2)}dt \\
&v_{yn} = v_{y(n-1/2)} + g\frac{dt}{2}, \quad n = 1, 2, \ldots
\end{aligned}$$

V smeri osi x je gibanje enakomerno zato se pot v smeri osi x na vsakem časovnem koraku dt poveča za $dx = v_{x0}dt$. V smeri osi y se hitrost s časom spreminja. Zato najprej zaradi natančnosti izračunamo hitrost na sredini časovnega intervala dt nato pa določimo še hitrost na koncu časovnega intervala. Zaradi lastnosti kvadratne funkcije, ta metoda določa trajektorijo poševnega meta natančno.

Procedura `top`

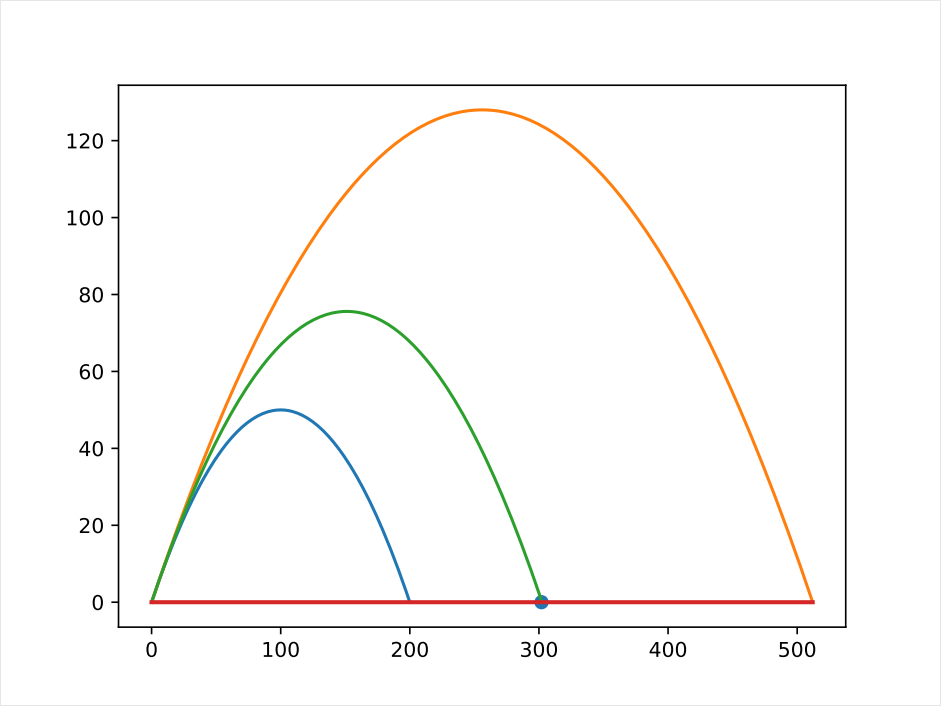
```
import random
p = random.randint(264, 480)
def top(smodnik):
    x, y, g = 0, 0, -1
    dt = 0.1
    vx = vy = smodnik
    while y >= 0:
        x += vx*dt
        vy += g*dt/2
        y += vy*dt
        vy += g*dt/2
    return x
```

Primer

Izhod:

```
307 338 13.00
307 264 11.50
307 300 12.25
307 319 12.62
307 309 12.44
307 304 12.34
307 307 12.39
```

Slika



5. **Sekantna metoda** Hitrejša in tudi bolj nevarna metoda za iskanje ničel funkcije je *sekantna metoda*. Medtem, ko metoda bisekcije zagotavlja pri izpolnjevanju pogojev tudi konvergenco metode, pri sekantni metodi ni tako preprosto.

   Pri tej metodi potrebujemo dva začetna približka a in b. Tu ni pogoja, da morata biti funkcijski vrednosti $f(a)$ in $f(b)$ različnega znaka. Skozi točki $(a, f(a))$ in $(b, f(b))$ položimo premico (*sekanto*) in poiščemo presečišče premice z osjo x. Premica seka os x v točki:

$$c = \frac{af(b) - bf(a)}{f(b) - f(a)}.$$

   Nato postavimo, $a = b$ in $b = c$ in izračunamo novo vrednost c. To počnemo toliko časa, da pade absolutna vrednost $|f(c)|$ pod predpisano vrednost ϵ.

6. **Metoda regula fasi** Metoda *regula falsi* ali tudi *metoda napačnega pravila*, poskuša odpraviti pomanjkljivost sekantne metode na račun nekoliko slabše konvergence. Ta metoda predpostavi, da se nahaja na intervalu $[a, b]$ natanko ena točka, kjer funkcija seka os x, in da sta funkcijski vrednosti v krajiščih a in b intervala različnega predznaka $f(a)\, f(b) < 0$. Pogoja sta enaka kot pri metodi bisekcije. Točko c izračunamo enako kot pri sekantni metodi:

$$c = \frac{af(b) - bf(a)}{f(b) - f(a)}$$

   Enako kot pri metodi bisekcije izberemo $a = c$ oziroma $b = c$ tako, da sta funkcijski vrednosti v krajiščih novega intervala $[a, b]$ še vedno različnega predznaka.

7. **Prepovedana tema**

Bralcem priporočamo, da to temo preskočijo.

Poglejmo natančneje, kako se izračuna vrednost c iz podatkov a, b in funkcije $f(x)$. Enačba premice skozi točki $(a, f(a))$ in $(b, f(b))$:

$$y - f(a) = \frac{f(b) - f(a)}{b - a}(x - a), \to y = 0, x = c, \to c = a - f(a)\frac{b - a}{f(b) - f(a)}.$$

Približajmo točko b k točki a. Pišemo, $b = a + h$, kjer je vrednost h majhna kolikor hočemo. Kako se v tem primeru zapiše gornja enačba? To bomo pogledali kar na konkretnem primeru. Naj bo:

$$\begin{aligned} f(x) &= x^3 - 3x + 1, \quad \text{in pišemo} \quad a = x, \quad b = x + h \to \\ \frac{f(x+h) - f(x)}{h} &= \frac{(x+h)^3 - 3(x+h) + 1 - x^3 + 3x - 1}{h} \\ &= \frac{3x^2h + 3xh^2 + h^3 - 3h}{h} = 3x^2 + 3xh + h^2 - 3. \end{aligned}$$

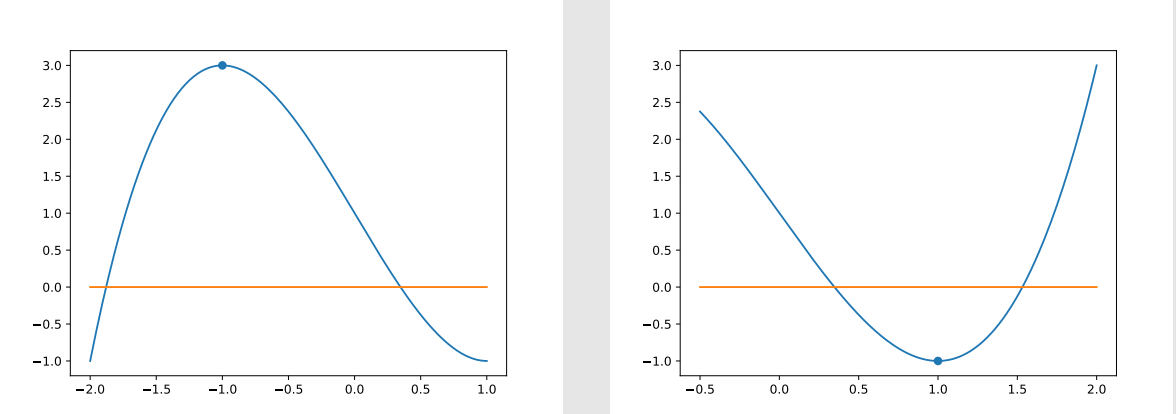
Ker je h poljubno majhen, ga postavimo kar na 0. Sedaj potrebujemo le en začetni približek x_0:

$$x_{n+1} = x_n - \frac{x_n^3 - 3x_n + 1}{3x_n^2 - 3}, \quad n = 0, 1, \ldots$$

S tem, ko smo približali b k a, smo lego sekante približali tangenti na graf funkcije v točki a. Zato se ta metoda imenuje *tangentna* ali *Newtonova* metoda. Metoda konvergira k rešitvi v splošnem hitreje kot prej omenjene metode. Vendar pa pri uporabi te metode moramo znati izračunat vrednost kvocienta, ko gre h proti nič. Ta vrednost pa je ravno odvod funkcije v točki a. Poznati moramo tudi dovolj natančno potek grafa funkcije v okolici ničle, da lahko pravilno izberemo začetno vrednost, tako da bo zaporedje $\{x_n\}$ konvergiralo k želeni ničli funkcije.

8. **Ekstremi funkcije**

Slika



Iščemo ekstrem funkcije $f(x)$ na intervalu $[a, b]$, to je vrednost neodvisne spremenljivke x, kjer funkcija $f(x)$ zavzame najmanjšo oziroma največjo vrednost. Predpostavimo, da je funkcija $f(x)$ zvezna (graf je neprekinjena krivulja) in da obstaja na intervalu $[a, b]$ natanko ena točka x_ϵ, kjer zavzame funkcija $f(x)$ najmanjšo oziroma največjo vrednost.

Nalogo rešujemo z metodo trisekcije. Interval $[a, b]$ razdelimo na tri enake dele s krajišči $[a, x_1, x_2, b]$, kjer je $x_1 = a + d$, $x_2 = b - d$ in $d = (b - a)/3$. Rešujemo po korakih. Na vsakem koraku popravimo krajišči intervala, tako da primerjamo funkcijski vrednosti v točkah x_1 in x_2. V primeru, da iščemo najmanjšo vrednost na intervalu v primeru, da je $f(x_1) < f(x_2)$ postane $b = x_2$, sicer pa $a = x_1$. Če pa iščemo največjo vrednost obrnemo neenačaj pri primerjanju funkcijskih vrednostih.

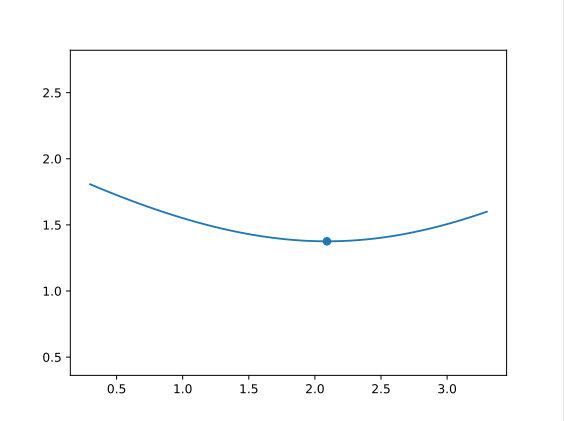
9. **Najkrajši čas**

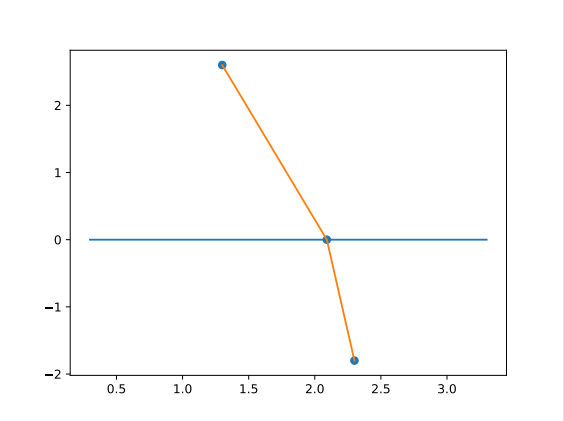
Kopalec na plaži se odpravi na otoček nedaleč od obale. Izbere pot, da bo porabil za pot najmanj časa. Hitrost hoje po kopnem $v_1 = 5.3$ km/h, medtem ko je hitrost plavanja $v_2 = 2.1$ km/h. Vzemimo, da obala teče po osi x. Zgornja polovica je kopno $y > 0$, medtem ko je spodnja polovica morje $y < 0$. Plavalec se odpravi iz točke $(x_1, y_1) = (2.3, 2.6)$ proti otoku v točki $(x_2, y_2) = (1.3, -1.8)$.

Na osi x določimo mesto x_0, ker bo kopalec zabredel v morje in odplaval proti otoku. Čas, ki ga bo porabil je

$$t = t_1 + t_2 = d_1/v_1 + d_2/v_2, \quad \text{kjer je} \quad d_1 = \sqrt{(x_1 - x_0)^2 + y_1^2}, \quad d_2 = \sqrt{(x_2 - x_0)^2 + y_2^2}.$$

Slika





Rešitve 1

Iteracija

```
#!/usr/bin/env python3
import matplotlib.pyplot as plt
import sympy, numpy as np

def f(a, x):
  return a * (x -1)  + 1


def pt(a):
    xh = 1
    yh = 1
    return (xh, yh)
```

```
if __name__ == '__main__':
  x0 = 0
  x1 = 2
  a = -1.2

  x = np.linspace(x0, x1, 200)
  y = [f(a,t) for t in x]

  plt.axes().set_aspect('equal')
  plt.xlim(x0, x1)
  plt.ylim(0, 2)

  xx = 0.7
  l = []
  l.append((xx,0))
  for _ in range(15):
    yy = f(a,xx)
    l.append((xx, yy))
    l.append((yy, yy))
    xx = yy
```

```
plt.plot(x, y)
plt.plot(x, x)
plt.plot([k[0] for k in l], [k[1] for k in l])
plt.plot([x[0], x[-1]], [0, 0])

xh, yh = pt(a)
plt.scatter([xh],[yh])

plt.show()
```

Logistična enačba

```
#!/usr/bin/env python3
import matplotlib.pyplot as plt
import sympy, numpy as np

def f(a, x):
  return a * x*(1 - x)

def df(a, x):
  return a - 2 * a * x

def pt(a):
    xh = (a - 1)/a
    yh = f(a, xh)
    return (xh, yh)

def lf(a, x):
    xh, yh = pt(a)
    return  df(a, xh)*(x - xh) + yh
```

```
if __name__ == '__main__':
  x0 = 0
  x1 = 1
  a = 1.2

  x = np.linspace(x0, x1, 200)
  y = [f(a,t) for t in x]

  plt.axes().set_aspect('equal')
  plt.xlim(x0, x1)
  plt.ylim(0, a/4 + 0.1)

  xx = 0.5
  ll = [0.5]
  l = []
  l.append((xx,0))
  for _ in range(30):
    yy = f(a,xx)
    ll.append(yy)
    l.append((xx, yy))
    l.append((yy, yy))
    xx = yy

  ly = lf(a, x)

  plt.plot(x, y)
  plt.plot(x, x)
  plt.plot([k[0] for k in l], [k[1] for k in l])
  plt.plot([x[0], x[-1]], [0, 0])
  plt.plot(x, ly)

  xh, yh = pt(a)
  plt.scatter([xh],[yh])
  plt.show()

  plt.plot(ll)
  plt.show()
```

Bisekcija

```python
#!/usr/bin/env python3
import matplotlib.pyplot as plt
import sympy, numpy as np

def f(x):
  return x**3 - 3*x + 1

def bisect(f, a, b):
  if f(a) * f(b) < 0:
    c = (a + b)/2
    while abs(f(c)) > 0.001:
      if f(a) * f(c) < 0:
        b = c
      else:
        a = c
      c = (a + b) / 2
  return c
if __name__ == '__main__':
  a = -1
  b = 1

  x = np.linspace(a, b, 200)
  y = [f(t) for t in x]
  c = bisect(f, a, b)
  print(c)

  plt.plot(x, y)
  plt.plot([x[0], x[-1]], [0, 0])
  plt.scatter([c], [0])
  plt.show()
```

Top

```
#!/usr/bin/env python3
import random
import matplotlib.pyplot as plt

p = random.randint(264, 480)

def bum(smodnik):
    x, y, g, dt = 0, 0, 1, 0.1
    xx, yy = [x], [y]
    vx = vy = smodnik
    while y >= 0:
        x += vx*dt
        vy -= g*dt/2
        y += vy*dt
        vy -= g*dt/2
        xx.append(x)
        yy.append(y)
    return x, xx, yy

def bisekcija():
    a, b = 10, 16
    while True:
        m = (a + b)/2
        q, xx, yy = bum(m)
        print('%d %d %0.2f' %(p, q, m))
        if abs(q - p) < 1:
            return m, xx, yy
        if q < p: a = m
        elif q > p: b = m
        else:
            return m, xx, yy
```

```
if __name__ == '__main__':
    q1, x1, y1 = bum(10)
    q2, x2, y2 = bum(16)
    q, x, y = bisekcija()
    plt.plot(x1, y1)
    plt.plot(x2, y2)
    plt.plot(x, y)
    plt.plot([0,x2[-1]],[0,0], linewidth=2)
    plt.scatter([p], [0])
    plt.show()
```

Sekantna

```
#!/usr/bin/env python3
import matplotlib.pyplot as plt
import sympy, numpy as np

def f(x):
  return x**3 - 3*x + 1

def sekantna(f, a, b):
  c = (a*f(b) - b*f(a))/(f(b) - f(a))
  res = [[a, a, c, c],[0, f(a),0, f(c)]]
  while abs(f(c)) > 0.001:
      a, b = b, c
      res.append([a, b])
      c = (a*f(b) - b*f(a))/(f(b) - f(a))
      res[0] += [c, c]
      res[1] += [0, f(c)]
  return c, res
```

```
if __name__ == '__main__':
  a = -1
  b = -0.5
  x = np.linspace(-2, 2, 200)
  y = [f(t) for t in x]

  c, res = sekantna(f, a, b)
  print(c)
  plt.plot(x, y)
  plt.plot([x[0], x[-1]], [0, 0])
  plt.scatter([c], [0])
  plt.plot(res[0], res[1])

  plt.show
```

Ragula falsi

```
#!/usr/bin/env python3
import matplotlib.pyplot as plt
import sympy, numpy as np

def f(x):
  return x**3 - 3*x + 1

def regula_falsi(f, a, b):
  res = []
  if f(a) * f(b) < 0:
    c = (a*f(b) - b*f(a))/(f(b) - f(a))
    while abs(f(c)) > 0.001:
      if f(a) * f(c) < 0:
        b = c
      else:
        a = c
      res.append([a, b])
      c = (a*f(b) - b*f(a))/(f(b) - f(a))
  return c, res
```

```
if __name__ == '__main__':
  a = -1
  b = 1.5
  x = np.linspace(a, b, 200)
  y = [f(t) for t in x]

  c, res = regula_falsi(f, a, b)
  print(c)
  plt.plot(x, y)
  plt.plot([x[0], x[-1]], [0, 0])
  plt.scatter([c], [0])
  for p in res:
      plt.plot([p[0],p[1]],[f(p[0]),f(p[1])])

  plt.show()
```

Tangentna

```python
#!/usr/bin/env python3
import matplotlib.pyplot as plt
import sympy, numpy as np

def f(x):
  return x**3 - 3*x + 1

def df(x):
  return 3*x**2 - 3

def tangentna(f, x_0):
  res = [[x_0, x_0],[0, f(x_0)]]
  x = x_0
  while abs(f(x)) > 0.001:
      x = x - f(x)/df(x)
      res[0] += [x, x]
      res[1] += [0,f(x)]
  return x, res
```

```python
if __name__ == '__main__':
  x = np.linspace(0, 3, 200)
  y = [f(t) for t in x]
  a = 1.1

  c, res = tangentna(f, a)
  print(c, res)
  plt.plot(x, y)
  plt.plot([x[0], x[-1]], [0, 0])
  plt.scatter([c], [0])
  plt.plot(res[0], res[1])
  plt.show()
```

Ektremi funkcije

```
#!/usr/bin/env python3
import matplotlib.pyplot as plt
import sympy, numpy as np

def minimum(f, a, b):
  d = b - a
  while d > 0.000001:
      x1, x2 = a + d / 3, b - d / 3
      y1, y2 = f(x1), f(x2)
      if y1 < y2: b = x2
      else: a = x1
      d = b - a
  return (a + b) / 2

def maximum(f, a, b):
  d = b - a
  while d > 0.000001:
      x1, x2 = a + d / 3, b - d / 3
      y1, y2 = f(x1), f(x2)
      if y1 > y2: b = x2
      else: a = x1
      d = b - a
  return (a + b) / 2

def extrem(f, a, b):
    c = (a + b) / 2
    if f(c) < f(a) and f(c) < f(b):
        return minimum(f, a, b)
    elif f(c) > f(a) and f(c) > f(b):
        return maximum(f, a, b)
    else:
        return None
```

```
if __name__ == '__main__':
    def f(x):
        return x**3 - 3*x + 1

    a, b = -0.5, 2
    x = np.linspace(a, b, 200)
    y = f(x)
    c = extrem(f, a, b)
    print(c)
    plt.plot(x, y)
    plt.plot([x[0], x[-1]], [0, 0])
    plt.scatter([c], [f(c)])
    plt.show()
```

Najkrajši čas

```python
#!/usr/bin/env python3

import numpy as np
import matplotlib.pyplot as plt
from extrem import *

par = ()

def time(T1, T2, v1, v2, x0):
    x1, y1 = T1
    x2, y2 = T2
    return np.sqrt((x1 - x0)**2 + y1**2)/v1 + np.sqrt((x2 - x0)**2 + y2**2)/v2

def f(x):
    T1, T2, v1, v2 = par
    return time(T1, T2, v1, v2, x)

if __name__ == '__main__':
    T1 = (1.3, 2.6)
    T2 = (2.3, -1.8)
    v1, v2 = 5.3, 2.1
    a = T1[0] - 1
    b = T2[0] + 1
    par = (T1, T2, v1, v2)

    x = np.linspace(a, b)
    y = f(x)
    c = extrem(f, a, b)

    plt.axis('equal')

    plt.scatter([c], [f(c)])
    plt.plot(x,  y)
    plt.show()

    plt.plot([a, b], [0, 0])
    plt.plot([T1[0],c,T2[0]], [T1[1],0,T2[1]])
    plt.scatter([T1[0],c,T2[0]], [T1[1],0,T2[1]])
    plt.show()
```