МАТЕМАТИЧКА ГИМНАЗИЈА

МАТУРСКИ РАД

из предмета Програмирање

Статистичко решавање варијационих проблема

Ученик:

Марк Можаровски, 4а

Ментор:

Теодор фон Бург

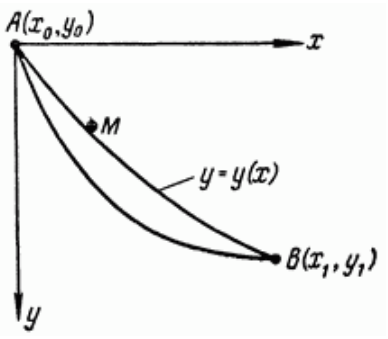
Београд, мај 2023.

Садржај

Увод

Брахистохрона, од грч. *βράχιστος* (најкраће) и *χρόνος* (време), представља криву најбржег спуштања. Проблем њеног налажења поставио је Јохан Бернули 1696. године овако:

Међу кривама које спајају тачке А и Б које се налазе у истој вертикалној равни (Б испод А), наћи ону криву крећући се по којој, под дејством само гравитације, материјална тачка стиже из тачке А у тачку Б за најкраће време (Слика 1).



Слика 1. Дефинисање брахистохроне

Решењем овог проблема бавили су се многи велики математичари и физичари, међу којима и Исак Њутн, Јакоб Бернули, Г. Б. Лајбниц, Г. Ф. Лопитал и Е. Б. Чирнхаус. Сви они, као и сам Јохан Бернули, решили су проблем на различите начине. Исак Њутн се, решавајући овај проблем, користио методом која је касније легла у основ варијационог рачуна.

Циљ овог рада је проналажење алгоритма уз помоћ ког би се, случајним променама положаја тачака неке криве, дошло до облика брахистохроне. У првом делу рада објашњене су методе којима ће се ово постићи, док је у другом приказан сам процес решавања.

1. Методе решавања проблема брахистохроне

1.1 Варијациони рачун

Варијациони рачун представља математичку дисциплину која се бави налажењем одређених функција тако да нека друга вредност, зависна од њих, достигне своју екстремну вредност (минимум или максимум).

Термини којима ћемо се користити:

- **Функционал**

Функционал $\Phi[f]$ представља функцију чији је домен скуп или простор функција, а кодомен скуп реалних или комплексних бројева.

Функционал $\Phi[f]$ додељује свакој функцији f одређен број из његовог кодомена.

- **Варијација**

Варијација $\delta\Phi$ представља аналог диференцијала у варијационом рачуну. За њу важи:

$$\delta\Phi = \Phi[f + \delta f] - \Phi[f].$$

Варијација $\delta\Phi$ је такође функционал, зато што зависи од f.

- **Варијациони извод**

Ако је могуће варијацију

$$\delta\Phi = \Phi[f + \delta f] - \Phi[f]$$

записати у облику

$$\delta\Phi = \int A(x)\delta f(x)dx$$

где је A нека функција од x, онда је A *варијациони извод* од Φ по f, у ознаци $\frac{\delta\Phi}{\delta f}$. Односно,

$$\frac{\delta \Phi}{\delta f} = A.$$

1.2 Ојлер-Лагранжова једначина

Задат је функционал $\Phi(f)$ од функције f, у зависности од променљиве x

$$\Phi(f) = \int_a^b F\big(x, f(x), f'(x)\big)dx,$$

на простору функција $f\colon [a,b] \to R$, где је f' ознака за први извод f по x. Ако функционал Φ достиже екстремум при некој функцији f, за њу онда важи обична диференцијална једначина:

$$\frac{\partial F}{\partial f} - \frac{d}{dx}\frac{\partial F}{\partial f'} = 0$$

која се зове ***Ојлер-Лагранжова једначина***.

Када члан $\frac{d}{dx}\frac{\partial F}{\partial f'}$ пребацимо на другу страну једначине, добијамо:

$$\frac{\partial F}{\partial y}_{|(x,y(x),y\prime(x))} = \frac{d}{dx}\left[\frac{\partial F}{\partial p}_{|(x,y(x),y\prime(x))}\right],$$

где је p ознака за f', а $\frac{\partial F}{\partial p}_{|(x,y(x),y\prime(x))}$ парцијални извод F по p. Рачунањем вредности овог извода добијамо функцију у зависности од (x,y,p) и њену вредност у тачки $(x,y(x),y'(x))$. Овим процесом добијамо функцију која је у зависности само од x.

1.3 Добијање једначине брахистохроне

Запишимо закон одржања енергије за материјалну тачку *M*:

$$\frac{mv^2}{2} = mgy,$$

где су
m – маса тела,
g – гравитацијоно убразање,
y – ордината,
v – брзина кретања тела.

Добијамо

$$v = \sqrt{2gy}\,.$$

Одатле се може наћи пројекција брзине на *x*-осу:

$$v_x = \frac{v}{\sqrt{1 + (y')^2}} = \frac{\sqrt{2gy}}{\sqrt{1 + (y')^2}}.$$

Време потребно за спуштање тачке *M* дуж криве одређене тачкама чије су ординате a и b је

$$\int_a^b \frac{1}{v_x}\,dx\,.$$

Отуд, проналажење најкраћег времена потребног за спуштање тачке *M* низ задату криву своди се на минимизацију вредности интеграла

$$T = \frac{1}{\sqrt{2g}} \int_0^L \sqrt{\frac{1 + (y')^2}{y}}\,dx$$

који смо добили убацивши вредност брзине v_x у интеграл добијен у претходном кораку, с тим што је као ордината прве тачке, ради лакшег рачуна, изабран координатни почетак.

Дакле, сада је потребно међу свим функцијама $y(x)$ које задовољавају границе $y(0) = 0$ и $y(L) = L$ наћи ону за коју Т достиже најмање могуће вредности, што ћемо постићи убацивањем интеграла *Т* у *Ојлер-Лагранжову једначину*.

$$F(x, y, p) = \sqrt{\frac{1 + p^2}{y}},$$
$$\frac{\partial F}{\partial p} = \frac{p}{\sqrt{y(1 + p^2)}}.$$

Рачунски добијамо једначине

$$\frac{d}{dx}\left[\frac{\partial F}{\partial p}_{|(x,y(x),y'(x))}\right] = \frac{1}{\sqrt{y(1 + (y')^2)}}\left(\frac{y''}{1 + (y')^2} - \frac{1}{2}\frac{(y')^2}{y}\right)$$

и

$$\frac{\partial F}{\partial y}_{|(x,y(x),y'(x))} = -\frac{\sqrt{1 + (y')^2}}{2y^{3/2}}.$$

Изједначавањем ове две једначине добијамо израз

$$y'' = -\frac{1 + (y')^2}{2y},$$

што је нелинеарна једначина. Њеним решавањем добићемо једначину најкраћег времена $y(x)$.

Први корак у решавању је свођење добијеног израза

$$2yy'' + 1 + (y')^2 = 0$$

на једначину првог реда. Помножимо обе стране са y':

$$2yy'y'' + y' + (y')^3 = 0.$$

Како је

$$2yy'y'' + y' + (y')^3 = [y + y(y')^2]' = 0,$$

закључујемо да је

$$y(1 + (y')^2) = C.$$

Одатле добијамо:

$$y' = \frac{dy}{dx} = \sqrt{\frac{C-y}{y}},$$

те важи

$$\frac{dx}{dy} = \sqrt{\frac{y}{C-y}}.$$

Уведимо смену $\frac{dx}{dy} = \operatorname{tg}\varphi$:

$$\frac{y}{C-y} = \frac{sin^2\varphi}{cos^2\varphi},$$

$$y = ycos^2\varphi + ysin^2\varphi = Csin^2\varphi.$$

Диференцирањем овог израза по φ долазимо до једначине

$$\frac{dy}{d\varphi} = 2C\sin\varphi\cos\varphi.$$

Аналогно, добијамо

$$\frac{dx}{d\varphi} = \sqrt{\frac{y}{C-y}}\frac{dy}{d\varphi} = 2Csin^2\varphi = C(1-\cos 2\varphi).$$

Интегралним рачуном, ово се може свести на

$$x = C\int(1-\cos 2\varphi)d\varphi = \frac{C}{2}(2\varphi - \sin 2\varphi).$$

За $x = 0$ важи да је $y = 0$, те одатле следи да је и $\varphi = 0$. Добијамо да је тангента криве у координатном почетку вертикална линија.

Добијен резултат је

$$x = \frac{C}{2}(2\varphi - \sin 2\varphi), \qquad y = C(1-\cos 2\varphi).$$

Суштински добили смо једначине

$$x = ct - c\sin t$$

и

$$y = c - c\cos t,$$

које описују брахистохрону, где је c параметар који зависи од почетних тачака, а t угао чије су вредности ограничене са $y(0) = 0$ и $y(L) = L$.

2. Програмирање решења

2.1 Алгоритам случајних промена

Идеја алгоритма којим ћемо решити проблем брахистохроне обухвата више корака.

Први јесте да праву која спаја задате тачке *А* и *Б* поделимо на *n* једнаких делова, где је *n* неки број у скупу природних бројева *N*.

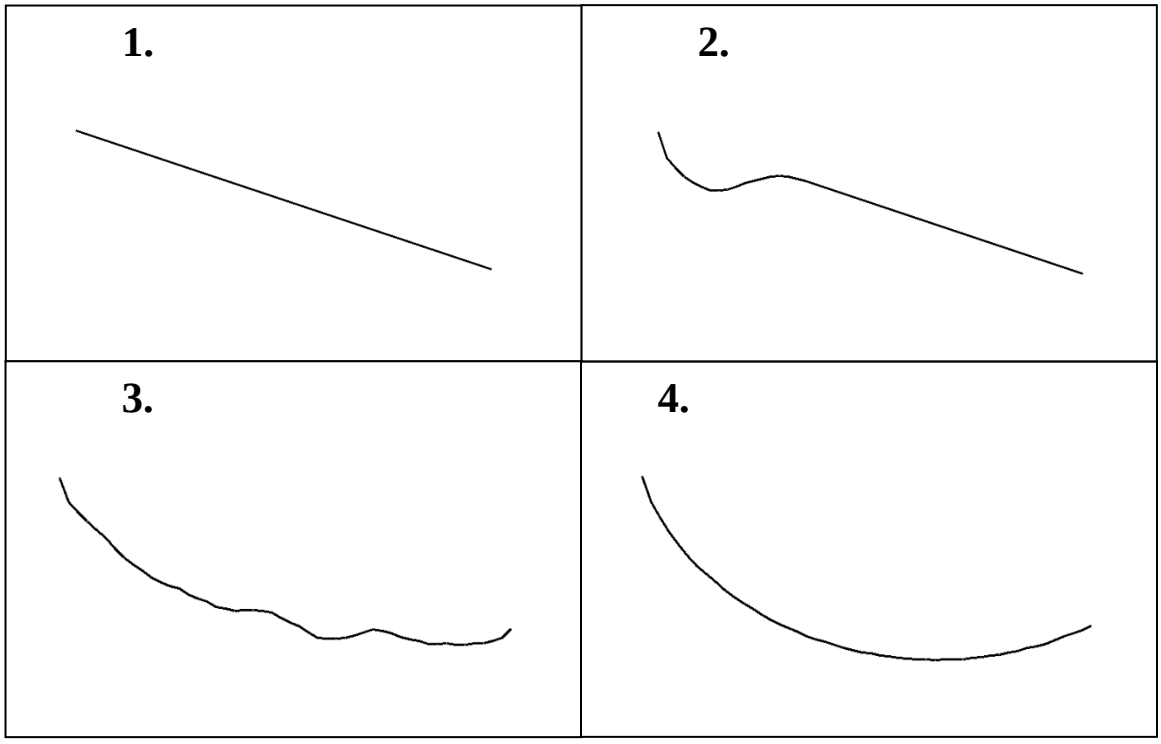
Затим, ту праву мењамо тако што ћемо променити ординате једне или више њених тачака, те по формули

$$\sum_{i=1}^{n-1} \frac{-v_i + \sqrt{v_i^2 + 2gl_i}}{g}$$

срачунати време потребно за спуштање материјалне тачке добијеном кривом.

У случају да је дошло до повећања овог времена, та промена се одбацује. Ако је израчунато време мање од времена добијеног пре нове промене, она се задржава.

Овај поступак се врши док се не добије жељена крива (Слика 2).



Слика 2. Алгоритам случајних промена

У овом случају, домен функционала Ф су све могуће криве f које спајају тачке *А* и *Б*, док је његов кодомен скуп реалних бројева и представља време потребно за спуштање материјалне тачке низ ту криву.

Случајну промену на ове криве означићемо са δf. Одатле важи да је $\delta\Phi$ варијација

$$\delta\Phi = \Phi[f + \delta f] - \Phi[f].$$

Услов за задржавање неке промене криве биће $\delta\Phi < 0$, односно $\Phi[f + \delta f] < \Phi[f]$.

2.2 Основа програма

За извршавање програма који се служи овим алгоритмом користимо Visual Studio 2022, због погодности које пружа приликом израде програма који се заснивају на графичком приказу неких вредности.

Креирамо Windows Forms Application пројекат у језику C#. Затим правимо методу која ће направити праву одређене висине *h* и дужине *l* и поделити је на *n* једнаких делова.

```
double l, h;

int n;

double[] x = new double[1000];
double[] y = new double[1000];

private void Reset()
{
    x[0] = 100;
    y[0] = 100;

    for (int i = 1; i < n; i++)
    {
        x[i] = x[i - 1] + l / n;
        y[i] = y[i - 1] + h / n;
    }
}
```

Метода *Form1_Paint* исцртава у покренутој апликацији тражену праву. Касније, користиће се и за цртање добијених крива.

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = CreateGraphics();
    Pen pen = new Pen(Color.Black, 3);
    for (int i = 0; i < n - 1; i++)
        g.DrawLine(pen, (int)x[i], (int)y[i], (int)x[i + 1], (int)y[i + 1]);
}
```

Следећи корак је написати методу која ће рачунати време потребно за спуштање неке материјалне тачке претходно добијеном кривом. Приметимо такође да ће тачке по *x*-оси остати на истом месту, јер им мењамо само *y* координате. Зато као параметар ова метода узима скуп *f* и враћа време ***t.*** Време потребно за спуштање се рачуна по формули:

$$\sum_{i=1}^{n-1} \frac{-v_i + \sqrt{v_i^2 + 2gl_i}}{g},$$

где је

$v_i = \sqrt{2g(y_{i-1} - y_0)}$ брзина на почетку i-тог одсечка,

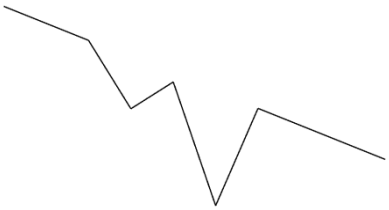
$l_i = \sqrt{(y_i - y_{i-1})^2 + (x_i - x_{i-1})^2}$ дужина i-тог одсечка,

$g = 9{,}81$ гравитационо убрзање.

```
private double Simulate(double[] f)
{
    double t = 0;
    for (int i = 1; i < n; i++)
    {
        double v = Math.Sqrt(2 * 9.81 * (f[i - 1] - f[0]));
        double l = Math.Sqrt((f[i] - f[i - 1]) * (f[i] - f[i - 1]) + (x[i] - x[i - 1]) * (x[i] - x[i - 1]));
        t += (-v + Math.Sqrt(v * v + 2 * 9.81 * l)) / 9.81;
    }
    return t;
}
```

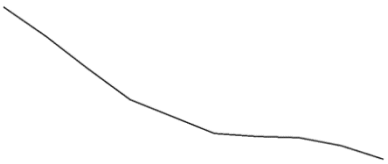
2.3 Промене криве

Криву можемо мењати тако што ћемо мењати ординате одвојених, насумично изабраних тачака, али у том случају је мала вероватноћа да ће после промене добијена крива бити глатка (Слика 3), што очигледно не одговара траженом облику брахистохроне (Слика 1).



Слика 3. Крива након промене ордината насумично одабраних, одвојених тачака

Глатку криву добијамо тако што ћемо одабрати неку тачку y_i и померити је за неки коефицијент α. Затим, тачке T_{i-1} и T_{i+1} померамо за неко α/exp. Тај процес наставимо тако померајући тачке $T_{i\pm k}$ за α/exp^k, све док је $\alpha/exp^k > 1$ (Слика 4).



Слика 4. Глатке промене криве

Коефицијенту α присвајамо случајан природан број из интервала одређеног променљивим *value* и *random.* За вредност exp узимамо реалан број већи од 1.

Дакле, креирана метода која мења облик криве за параметре узима променљиве *i, value, random* и *exp*. У сврху каснијег коришћења методе, враћа вредности **true** или **false,** у зависности од тога да ли је $\delta\Phi < 0$ или $\delta\Phi \geq 0$. Низ p је помоћни низ у који записујемо промењене вредности, да би после, користећи методу *Simulate***,** упоредили време потребно за спуштање тачке низ новонасталу криву.

```
private bool Tweak(int i, int v, int r, double e)
{
    a = v + rand.Next(-r, r);
    p[0] = y[0]; p[n - 1] = y[n - 1];
    for (int j = 0; j < n - 1; j++)
    {
        if (i - j > 0)
            p[i - j] = y[i - j] + a;
        if (i + j < n - 1)
            p[i + j] = y[i + j] + a;

        a /= e;
    }

    if (Simulate(p) < s)
    {
        for (int w = 0; w < n; w++)
            y[w] = p[w];

        s = Simulate(y);
        return true;
    }
    return false;
}
```

2.4 Интерполација

Интерполација представља веома брз и веома прецизан алгоритам налажења брахистохроне. Алгоритам чијом креацијом се бави овај рад користиће се методом интерполације ради провере прецизности, тј. упоређиваће добијену криву са оном која се добије методом интерполације. Овај процес детаљније је описан у следећем делу 2.5, а сада следи опис самог алгоритма интерполације.

Узмемо једначине добијене у делу 1.3:

$$x = ct - c\sin t\,, \qquad y = c - c\cos t.$$

За дате координате почетних тачака $(x_0, y_0), (x_1, y_1)$ узмемо да је $l = x_1 - x_0$ и $h = y_1 - y_0$. Онда важи:

$$\frac{l}{h} = \frac{t - \sin t}{1 - \cos t}.$$

Како бисмо пронашли вредност t за коју важи ова једначина, користимо алгоритам бинарне претраге, где је почетна вредност $t = \pi$. Добијену вредност враћамо у једначину:

$$c = \frac{h}{1 - \cos t}.$$

```
double c, t0;
t0 = Math.PI;
for (double j = 1; true; j++)
{
    if ((t0 - Math.Sin(t0)) / (1 - Math.Cos(t0)) > l / h)
        t0 -= Math.PI / Math.Pow(2, j);
    else if ((t0 - Math.Sin(t0)) / (1 - Math.Cos(t0)) < l / h)
        t0 += Math.PI / Math.Pow(2, j);
    else
    {
        c = h / (1 - Math.Cos(t0));
        break;
    }
}
```

Затим, за тачке од T_1 до T_{n-2}, користећи опет алгоритам бинарне претраге, налазимо прво t_i које задовољава

$$c(t_i - \sin t_i) = x_i - x_0,$$

те y_i рачунамо као

$$y_i = y_0 + c(1 - \cos t).$$

```
for (int i = 1; i < n - 1; i++)
{
    double t = Math.PI;
    for(double j = 1; true; j++)
    {
        if ((int)(c * (t - Math.Sin(t))) == (int)(x[i] - x[0]))
        {
            y[i] = y[0] + c * (1 - Math.Cos(t));
            break;
        }

        if ((int)(c * (t - Math.Sin(t))) > (int)(x[i] - x[0]))
            t -= Math.PI / Math.Pow(2, j);

        else
            t += Math.PI / Math.Pow(2, j);

    }
}
```

2.5 Провера прецизности

Због тога што нам интерполација даје прецизније резултате, скуп ордината тачака који је добијен овим методом назваћемо $yPerfect$. Тражени алгоритам проверава прецизност са којом су добијени положаји тачака нове криве методом квадратног одступања по формули

$$Accuracy = 100 - \frac{100}{n-2}\sum_{i=1}^{n-2}\frac{(y_i - yPerfect_i)^2}{\left(i \cdot {}^{h}/_{n} - yPerfect_i\right)^2}.$$

Фактор $\left(i \cdot {}^{h}/_{n} - yPerfect_i\right)^2$ је потребан како бисмо вредности прецизности добијали у интервалу од 0 до 100. Сума се рачуна од 1 до $n-2$ због фиксираности прве и последње тачке криве.

```
private double CheckAccuracy(double[] y)
{
    double accuracy = 100;

    for (int i = 1; i < n - 1; i++)
            accuracy -= (y[i] - yPerfect[i]) * (y[i] - yPerfect[i])
            / (i * h / n - yPerfect[i]) / (i * h / n - yPerfect[i])
            / (double)(n - 2) * 100.00;

    return accuracy;
}
```

Очекујемо да за наш алгоритам са порастом броја итерација, тј. промена координата тачака, расте и прецизност. На Графику 1 приказана је зависност прецизности добијене криве од броја итерација за вредности $n = 50$ и параметара $value = 12,\ random = 16,\ exp = 1.1$.

На y-оси је означена прецизност, а на x-оси број итерација (један подеок на x-оси представља пет итерација). Овај график одговара просечно двадесет пуштених програма. Интерполацијом се брже достиже жељена прецизност (99%).

Проверимо шта ће се десити ако значајно повећамо параметре на $value = 30$, $random = 40,\ exp = 1.1$. Са Графика 2 се види да се за веће вредности параметара значајно брже достиже већа прецизност, али после одређеног тренутка брзина са којом расте прецизност опада. Анализирајући зашто долази до таквог понашања долази се до закључка да након тренутка у ком прецизност постане већа од 98% крива постаје веома близу идеалној, са пар мањих грешака. Сваки следећи пут, због великих вредности $value$ и $random$, долази до „прескакања“ идеалних координата тачака.

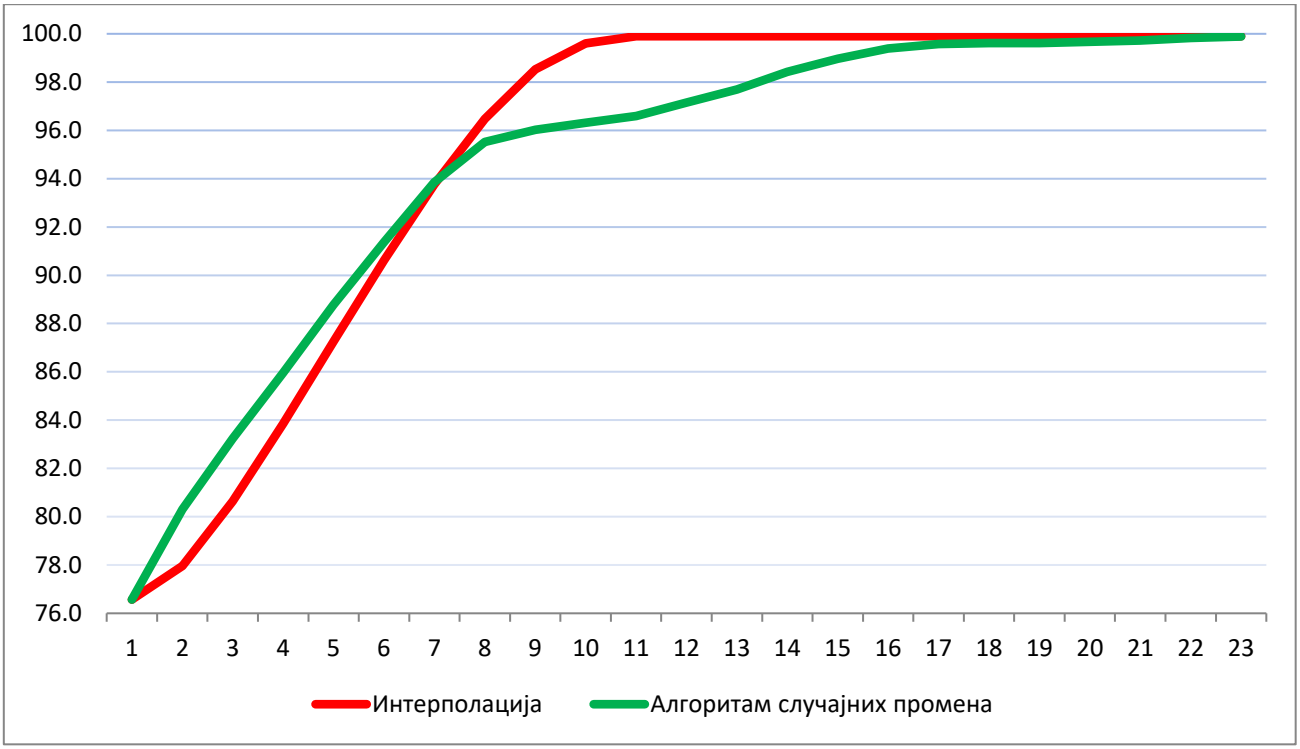


График 1.

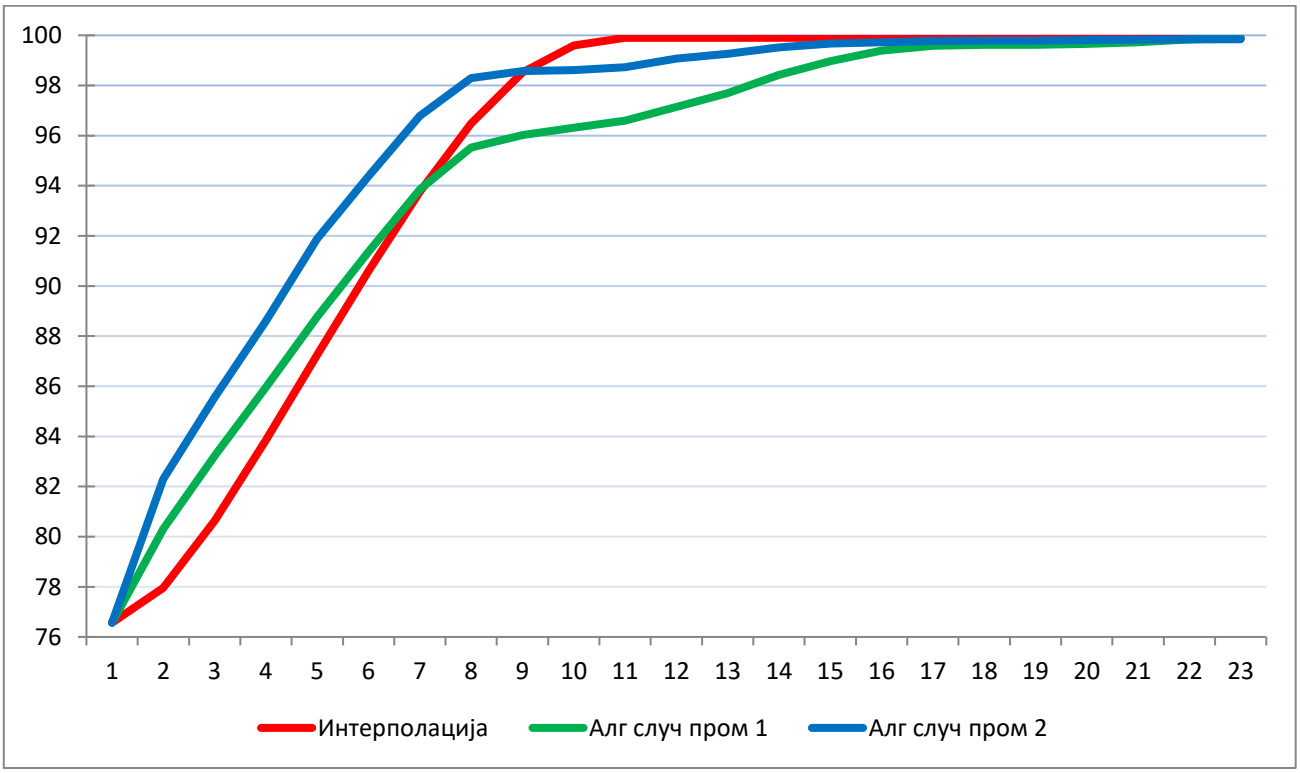


График 2.

Ради избегавања "прескакања" ћемо након достигнуте прецизности од 98% променити вредности параметара на $value = 4,\ random = 10,\ exp = 1.14$.

На Графику 3 је на x-оси означен тачан број итерација. Овај пут програм је пуштен 200 пута.

Када увећамо део Графика 3 где је вредност прецизности између 98% и 100% (График 4), видимо да се жељена прецизност од 99% постиже брже алгоритмом случајних промена него методом интерполације, што је и био циљ.

Не желимо превише да повећамо вредности параметара, јер ће онда долазити до превише великих промена за које је велика вероватноћа да неће доводити до смањења времена спуштања тачке кривом. Добијање брахистохроне ће се свести на игру случаја са набадањем одговарајуће почетне тачке и случајне вредности α.

У супротном, ако задајемо премале вредности параметара, промене ће бити високе прецизности, односно скоро сваки пут ће се време потребно материјалној тачки за спуштање низ криву смањивати. Због овога ће, пак, број потребних итерација бити превише велик. Такав поступак би, у поређању са бинарном претрагом, давао значајно лошији резултат.

Цела идеја алгоритма и уопште разлог због ког је бржи од алгоритма интерполације је померање неколико тачака одједном. Зато је веома битно како одаберемо параметар exp. За велике вредности померамо мање тачака, али је шанса успешне промене већа, док за мале вредности померамо више тачака одједном, што углавом доводи до бржег добијања брахистохроне.

Не постоји једна одређења вредност ни за један од параметара која ће давати најбољи резултат, већ се алгоритам своди на проналажење статистички највише одговарајућих параметара.

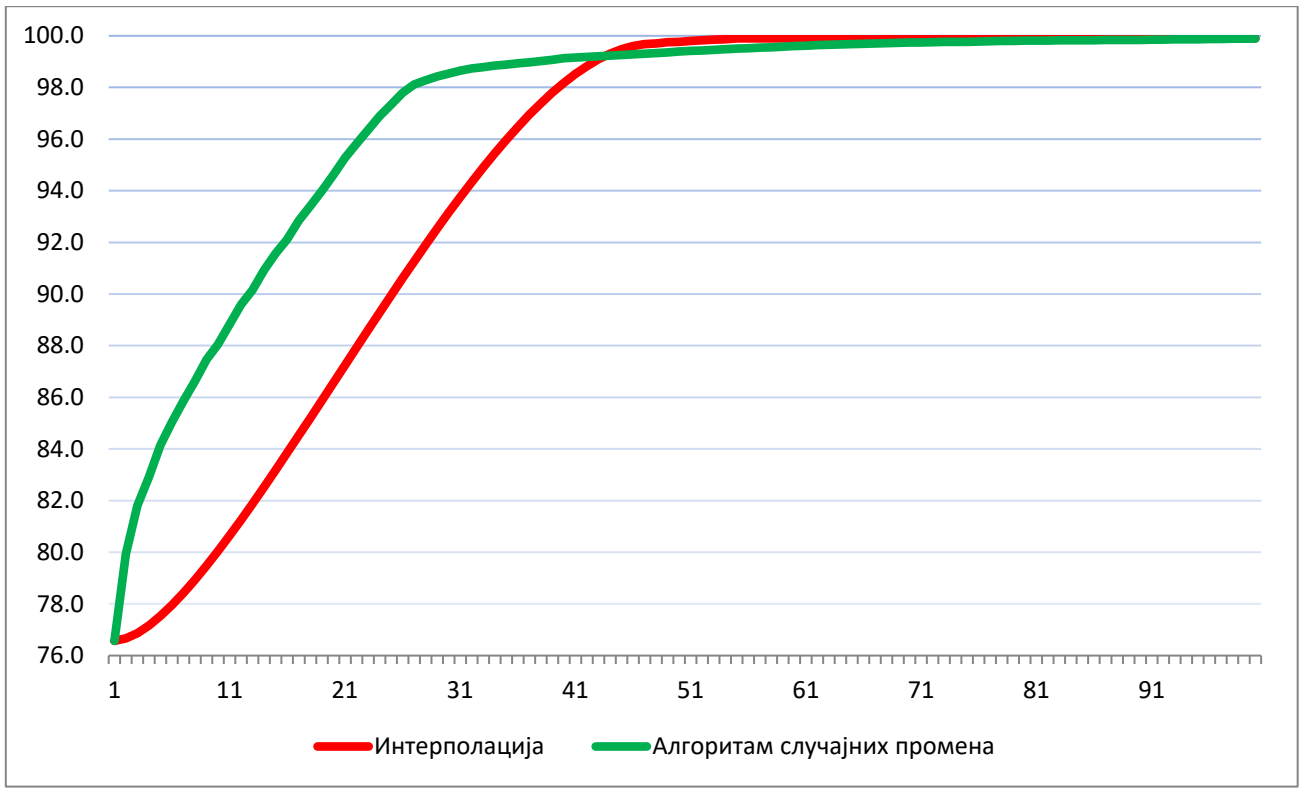


График 3.

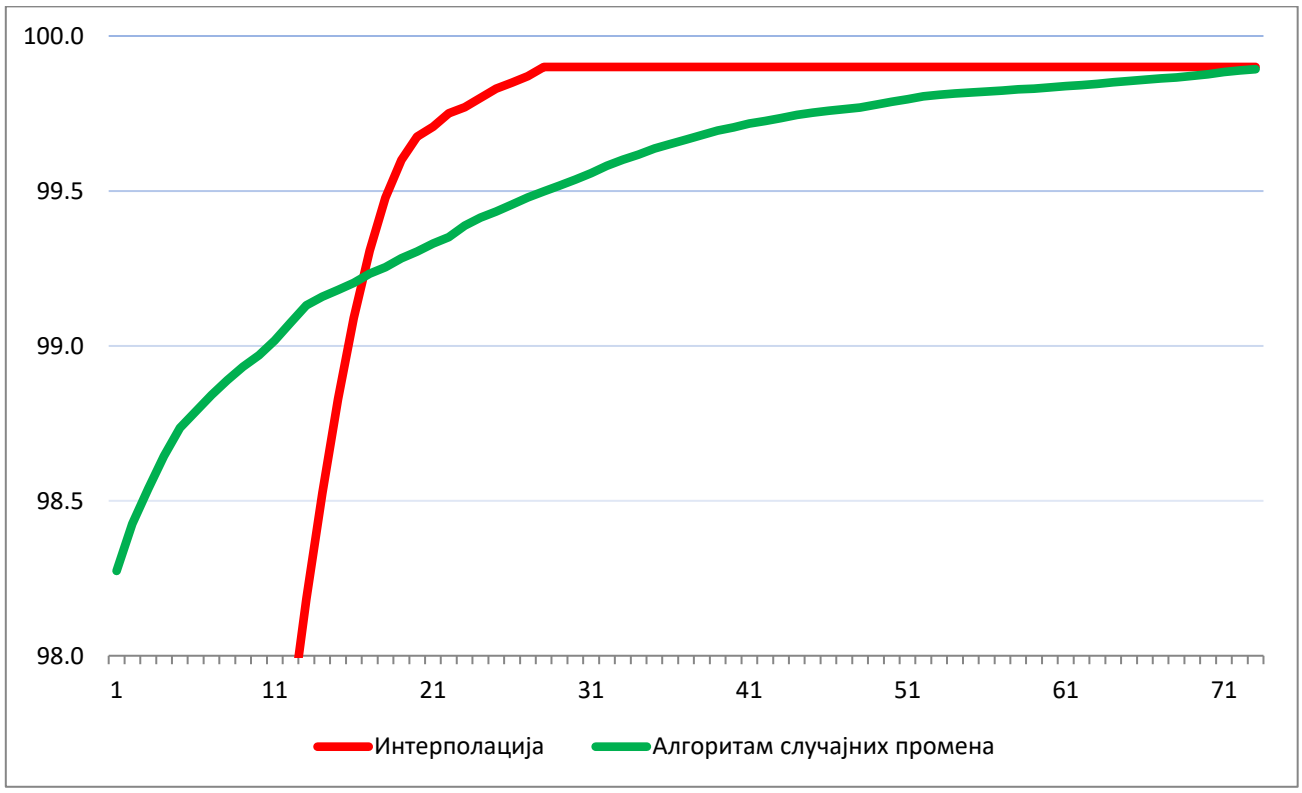


График 4.

Закључак

Варијациони проблеми, почевши од проблема налажења брахистохроне крајем 17. века, своју примену већ стотинама година налазе у различитим математичким и физичким дисциплинама. Њиховим решавањем, више или мање успешно, бавили су се различити научници и инжењери. Данас је, са друге стране, познат низ алгоритама и рачунарских програма уз помоћ којих се лако долази до њихових решења.

Алгоритам случајних промена, којим се овај рад бави, служи се методама које могу бити корисне у различите сврхе. Поред проблема брахистохроне, на основу њега долази се до решења и других варијационих проблема.

Посебно је користан при решавању комплекснијих проблема овог типа, где је потребно решити више једначина како би се дошло до крајњег одговора. Док би друге методе захтевале више времена и компликованијих измена у самом програму, при коришћењу алгоритма случајних промена потребно је једноставно додати нове параметре и касније проверити прецизност нађеног решења. Како, захваљујући овој особини, не тражи знање више математике или диференцијалног рачуна, приступачан је и за ширу употребу. Уз тачно подешавање коришћених параметара постиже значајно бржа и прецизнија решења у односу на друге, сличне алгоритме.

Литература

1. Paul J. Nahin, “When Least is Best”, *Princeton U.P.*, 2004.
2. Алексеев В. М., Тихомиров В. М., Фомин С. В., “Оптимальное управление”, *Главная редакция физико-математической литературы,* 1979.
3. Clegg, J.C., “Calculus of Variations”, *Interscience Publishers Inc*., 1968.
4. https://encyclopediaofmath.org
5. https://mathworld.wolfram.com
6. https://planetmath.org