In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import matplotlib.lines as lns

plt.rcParams['figure.figsize'] = [12, 12]
plt.rcParams['figure.dpi'] = 100

def plot_function(interval,fun):
    a=interval[0]
    b=interval[1]

    x=np.linspace(a,b,100)
    y1=fun(x)

    plt.figure(figsize=(15, 10))
    plt.plot(x,y1,linewidth=5)
    plt.plot(x,np.zeros(x.size),linewidth=5)


def draw_vertical_lines(a,b):
    l1=lns.Line2D([a[0],a[0]],[0,a[1]],color="blue",linewidth=5)
    l2=lns.Line2D([b[0],b[0]],[0,b[1]],color="blue",linewidth=5)
    ax = plt.gca()
    ax.add_line(l1)
    ax.add_line(l2)

def plot_sample(a,b,fun,N):
    xi = np.random.rand(N)*(b-a)+a
    ax = plt.gca()
    for i in range(N):
        ln=lns.Line2D([xi[i],xi[i]],[0,fun(xi[i])],color="red",linewidth=3)
        ax.add_line(ln)

def plot_rectangle(a,b,x,fun):
    ax = plt.gca()
    rect = patches.Rectangle((a, 0),abs(b-a),fun(x),facecolor="blue")
    ax.add_patch(rect)

def calculate_error(a,b,fun,max_N,correct_solution):
    reps=np.arange(1000,max_N+1,50)

    num_reps=len(reps)
    errors=np.zeros(num_reps)
    results=np.zeros(num_reps)

    pos=0

    for i in range(num_reps):
        xi=np.random.rand(reps[i])*(b-a)+a
        f_avg=np.mean(fun(xi))
        I=(b-a)*f_avg
        results[pos]=I
        errors[pos]=np.abs(I-correct_solution)
        pos=pos+1

    plt.plot(reps,results,linewidth=6)
    plt.xlabel('Velicina uzorka')
    plt.ylabel('Rezultat integracije')

    ln=lns.Line2D([0,max_N], [correct_solution,correct_solution],linewidth=5,color="red")
    ax = plt.gca()
    ax.add_line(ln)

    return [errors,reps]
```

Monte Karlo integracija

Pomoću numeričke integracije možemo da odredimo određeni integral proizvoljne funkcije.

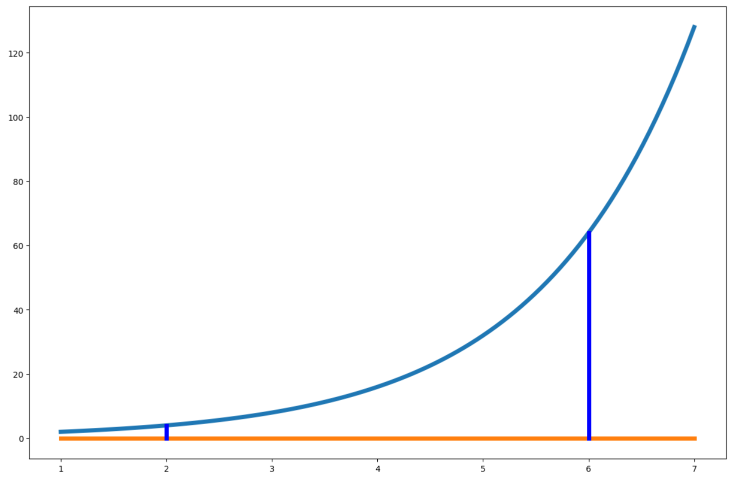
Određeni integral predstavlja površinu figure ispod date funkcije na zadatom zatvorenom intervalu.

Na primer, na sledećoj slici određeni integral

$$\int_2^6 2^x dx$$

je površina figure ispod funkcije $f(x) = 2^x$ na zatvorenom intervalu [2,6].

In [2]:
```
plot_function([1,7],lambda x:2.**x)
draw_vertical_lines([2.,2.**2],[6.,2.**6])
```



Ranije smo učili Njutn-Kotesove metode koje funkcionišu na sledeći način:

Aproksimiramo $f(x)$ fukcjiom $g(x)$, pa integralimo $g(x)$:

metoda trapeza - $g(x)$ je prava

Smipsonova metoda - $g(x)$ je parabola,....

Pored toga učili samo i Rombergovu integraciju, i Gausovu kvadraturu.

Ove metode su generalno jednostavne za impelmentaciju i imaju dobru tačnost.

Međutim, ne predstavljaju dobro rešenje za višestruke integrale:

$$\int\cdots\int_V u(u_1,\ldots,u_k)\,du_1\ldots du_k$$

Vištestruki integrali pojavljuju se recimo u kompujterskoj grafici, najviše kada se određuje osvetljenje u prostoru koji treba da se renderuje. Određivanje centra mase tela potrebno za simulaciju fizike takođe je problem koji formulisan kao višestruki integral.

Objasnićemo ukratno šta su višestruki integrali. Nakon toga pokazaćemo zašto su Njutn-Kotesove metode loše za rešavanje ovakvih integrala. Posle toga ćemo pokazati Monte Karlo metodu kao dobru alternativu.

Pre nego što nastavimo, napomenućemo da su sve metode koje smo do sada učili determinstičke, više primena istog algoritma na isti problem uvek daju isti rezultat.

Višestruki integraili - ukratko

Višestruki integral je vrsta određenog integrala na fukcijama koje imaju više promenljivih, na primer $f(x, y)$ ili $f(x, y, z)$.

Integrali funkcije sa dve promenljive nad nekim regionom u R^2 zovu se dvostruki integrali.

Kao što određeni integral pozitivne funkcije jedne promenljive predstavlja površinu ispod podintegralne funkcije na zadatom intervalu na x-osi, dvostruki integral pozitivne funkcije predstavlja zapreminu tela između površine definisane funkcijom $z = f(x, y)$ i tela definisanog domenom vrednosti x i y u $x - y$ ravni. Istu zapreminu mogli bi dobiti i pomoću trostrukog integrala konstantne funkcije $f(x, y, z) = 1$ nad trodimenzionalnim telom definisanog kao u prethodnoj rečenici.

Višestruki integrali za više od 3 promenljive su onda "zapremine" hiper-tela.

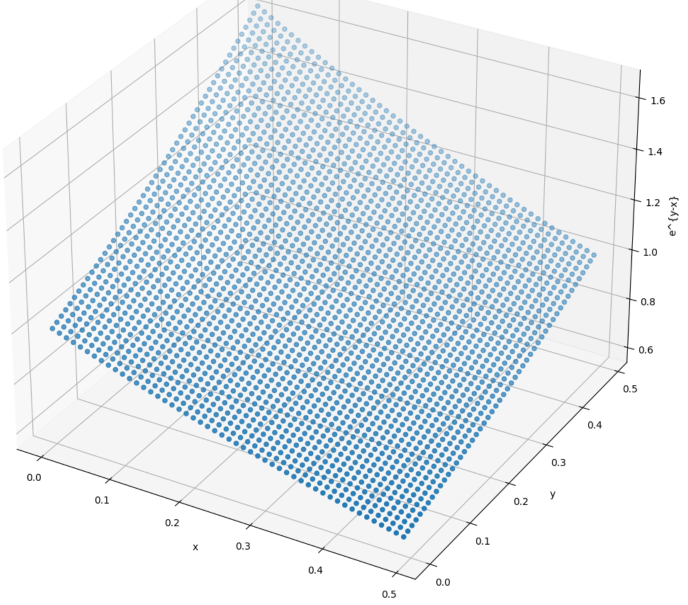
U nastavku pokazujemo kako izgleda grafik jednog višestrukog integrala in onda rešavamo taj integal pomoću metode trapeza za početak.

Prikazujemo i rešavamo sledeći dvostruki integral:

$$I = \int_0^{\frac{1}{2}}\int_0^{\frac{1}{2}} e^{y-x} dydx$$

In [3]:
```
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(15, 10))
ax = Axes3D(fig, auto_add_to_figure=False)
fig.add_axes(ax)

tx=ty=np.arange(0.,0.5,0.01)
xv, yv = np.meshgrid(tx, ty)
ax.scatter(xv,yv,np.exp(yv-xv))

ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_zlabel("e^{y-x}")
```

Out[3]:
```
Text(0.5, 0, 'e^{y-x}')
```



Koristimo metodu trapeza da rešimo odredimo vrednost integrala I.

Koristimo tri tačke za metodu trapeza na intervalu $[a, b]$:

$$I = \int_a^b f(x)dx \approx \frac{h}{2}\left[f(a) + 2f\left(\frac{a+b}{2}\right) + f(b)\right] h = \frac{b-a}{2}$$

Ako je $a = 0$, $b = \frac{1}{2}$ i $n = 2$ onda $h = \frac{(\frac{1}{2} - 0)}{2} = \frac{1}{4}$, tako da onda imamo:

$$I = \int_0^{\frac{1}{2}} f(x)dx \approx \frac{1}{8}\left[f(0) + 2f\left(\frac{1}{4}\right) + f(\frac{1}{2})\right]$$

Kako bi mogli da primenimo metodu trapeza, konvertovaćemo dvostruki integral u jednostruki na sledeći način. Definišemo funkciju $g(x)$:

$$g(x) = \int_0^{\frac{1}{2}} e^{y-x}dy$$

Onda integrali I definišemo na sledeći način:

$$I = \int_0^{\frac{1}{2}} g(x)dx$$

Primenjujemo sada metodu trapeza:

$$I = \int_0^{\frac{1}{2}}\int_0^{\frac{1}{2}} e^{y-x}dydx = \int_0^{\frac{1}{2}} g(x)dx \approx$$

$$\approx \frac{1}{8}\left[g(0) + 2g\left(\frac{1}{4}\right) + g(\frac{1}{2})\right]$$

$$\approx \frac{1}{8}\left[\int_0^{\frac{1}{2}} e^{y-0}dy + 2\int_0^{\frac{1}{2}} e^{y-\frac{1}{4}}dy + \int_0^{\frac{1}{2}} e^{y-\frac{1}{2}}dy\right]$$

Primenjujemo sada metodu trapeza na svaki od 3 integrala u prethodnom redu.

$$\int_0^{\frac{1}{2}} e^{y-0}dy \approx \frac{1}{8}\left[e^{0-0} + 2e^{\frac{1}{4}-0} + e^{\frac{1}{2}-0}\right]$$

$$\int_0^{\frac{1}{2}} e^{y-\frac{1}{4}}dy \approx \frac{1}{8}\left[e^{0-\frac{1}{4}} + 2e^{\frac{1}{4}-\frac{1}{4}} + e^{\frac{1}{2}-\frac{1}{4}}\right]$$

$$\int_0^{\frac{1}{2}} e^{y-\frac{1}{2}}dy \approx \frac{1}{8}\left[e^{0-\frac{1}{2}} + 2e^{\frac{1}{4}-\frac{1}{2}} + e^{\frac{1}{2}-\frac{1}{2}}\right]$$

Zamenjujemo prethodna tri integrala sada u postupak ranije:

$$\approx \frac{1}{8}\left[\int_0^{\frac{1}{2}} e^{y-0}dy + 2\int_0^{\frac{1}{2}} e^{y-\frac{1}{4}}dy + \int_0^{\frac{1}{2}} e^{y-\frac{1}{2}}dy\right]$$

$$\approx \frac{1}{8}\left[\frac{1}{8}\left[e^{0-0} + 2e^{\frac{1}{4}-0} + e^{\frac{1}{2}-0}\right] + 2\frac{1}{8}\left[e^{0-\frac{1}{4}} + 2e^{\frac{1}{4}-\frac{1}{4}} + e^{\frac{1}{2}-\frac{1}{4}}\right] + \frac{1}{8}\left[e^{0-\frac{1}{2}} + 2e^{\frac{1}{4}-\frac{1}{2}} + e^{\frac{1}{2}-\frac{1}{2}}\right]\right]$$

$$\approx \left[\frac{1}{64}\left[e^{0} + 2e^{\frac{1}{4}} + e^{\frac{1}{2}}\right] + \frac{1}{32}\left[e^{-\frac{1}{4}} + 2e^{0} + e^{\frac{1}{4}}\right] + \frac{1}{64}\left[e^{-\frac{1}{2}} + 2e^{\frac{1}{2}} + e^{0}\right]\right]$$

$$\approx 0.25791494889765$$

Kod prethodnog primera koristili smo po 3 tačke za metodu trapeza po svakoj dimenziji, što je ukupno 9 tačaka koje su prikazane u nastavku.

In [4]:

```
x_points = [0.,0.25,0.5]
y_points = [0.,0.25,0.5]

x, y = np.meshgrid(x_points, y_points)

np.stack((x,y),axis=2)
```

Out[4]:

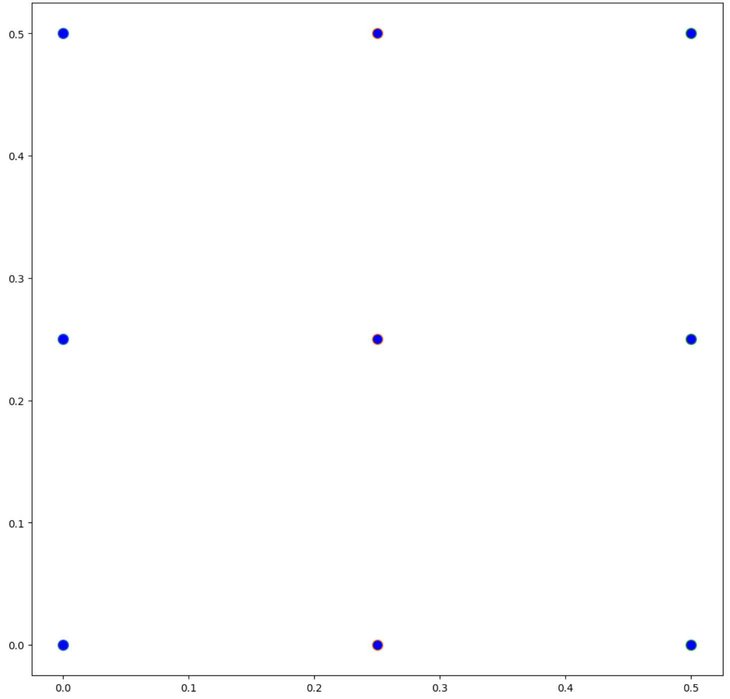
```
array([[[0.  , 0.  ],
        [0.25, 0.  ],
        [0.5 , 0.  ]],

       [[0.  , 0.25],
        [0.25, 0.25],
        [0.5 , 0.25]],

       [[0.  , 0.5 ],
        [0.25, 0.5 ],
        [0.5 , 0.5 ]]])
```

In [5]:

```
plt.plot(x,y,'o',markersize=10,markerfacecolor='b')
```

Out[5]:

```
[<matplotlib.lines.Line2D at 0x1e578a77ba8>,
 <matplotlib.lines.Line2D at 0x1e578a77c50>,
 <matplotlib.lines.Line2D at 0x1e578a77d30>]
```



Međutim, 3 tačke je veoma mali broj, takođe rešavali smo integral male dimenzionalnosti (dvostruki).

Sa porastom dimenzija i povećanjem broja tačka metode kao što je trapezna, Simpsonva itd. postaju nepraktične.

Na primer za petostruki integral i 10 tačaka po metodi ukupno nam je potrebno 10^5 tačaka.

Iz tog razloga potrebna nam je alternativna metoda koja je robusna na dimezionalnost (višestrukost) integrala.

Upravo takva alternativa je Monte Karlo metoda za numeričku integraciju.

Pre nego što detaljno objasnimo Monte Karlo integraciju ukratko ćemo prikazati Monte Karlo simulaciju koja čini širu oblast kojoj pripada Monte Karlo integracia.

Monte Karlo simulacija - ukratko

Monte Karlo simulacija oslanja se na ideju upotrebe nasumično generisanih vrednosti za rešavanje problema koji mogu da budu u principu i deterministički. Cilj je da se koristi ogroman broj nasumično izvršenih simulacija (eksperimenata) sa svrhom da će se po zakonu velikih brojeva pojaviti očekivani rezultat problema koji rešavamo.

Monte Karlo princip osmislio je Stanislav Ulam 1940-tih godina dok je radio u Los Alamos National Laboratory na projektu razvoja nuklearne bombe. Njegovu ideju iskoristio je John von Neumann koji je u okviru istog projekta napisao kod za računar ENIAC koji je vršio prve Monte Karlo simulacije.

Interesatno je da Stanislav Ulam osmislio MK princip dok se opravljao od bolesti i igrao Canfield solitaire igru sa kartama. Hteo je da proveri kolika je verovatnoća da će promenšan špil od 52 karte moći da se složi uspešno u okviru Canfield solitaire igre. Shvatio je da to nije tako lako rešiv problem koristeći metode iz oblasti kombinatorike. Ono što mu je palo na pamet kao rešenje je da se naparavi jednostavan algoritam za igranje Canfield solitaire igre na računaru i da se onda pokrene da odigra ogroman broj partija, i da se iz rezultata vidi kolika je verovatnoća uspešnog završetka igre. Suština je u tome da algoritam koji igra ne mora da bude jako sofisticiran već brz da bi mogao da se simulira veliki broj partija.

'Monty Hall' problem

Monte Karlo simulaciju demonstriraćemo na jednom interesantnom problemu oko koga su se vodile mnoge debate. Recimo da ste učesnik u nekom šou na televiziji. Šou je takav da postoje troje vrata. Iza jednih vrata je automobil, a iza preostalih vrata su koze. Učesnik bira vrata koja želi da otvori. Voditelj, koji zna šta je iza svih vrata, otvori neka druga vrata iza kojih je koza i onda pita učesnika da li želi da otvori vrata koja je prvi put birao ili da promeni vrata. Iako deluje kao da je verovatnoća jednaka, učesniku je bolje da promeni vrata.

Postoji mnogo rasprava na temu zašto je bolje zameniti vrata. Nećemo ulaziti u detalje. Pokazaćemo jedno tumačenje. Nakon toga ćemo pomoću Monte Karlo metode eksperimentalno potvrditi da je bolje zameniti vrata.

Jedno od tumačenja: Na početku učesnik ima verovatnoću $\frac{1}{3}$ da odabere vrata iza kojih je auto (pogledati slike ispod). Recimo da je odabrao prva vrata. Verovatnoća da je auto iza vrata 2 ili 3 je $\frac{2}{3}$. Recimo da voditelj otvori vrata 3 i da je iza njih koza. Tada vrata 2 imaju verovatnoću $\frac{2}{3}$ da je iza njih auto (pogledati slike ispod), pa je tada učesniku bolje da zameni vrata.

Mi ćemo u kodu simulirati veliki broj partija ove igre i iz ishoda proveriti da li je učesniku bilo bolje da zameni vrata ili ostane pri prvom izboru. Simulaciju smo realizovali tako da ako prvo nasumično rasporedimo auto i koze. Onda učesnik bira vrata. Ako je iza vrata koje je odabrao bio auto onda povećavamo broj ishoda kod kojih je bilo bolje da ostane pri prvom izboru, a u suprotnom povećavamo broj ishoda kod kojih je bilo bolje da zameni vrata.

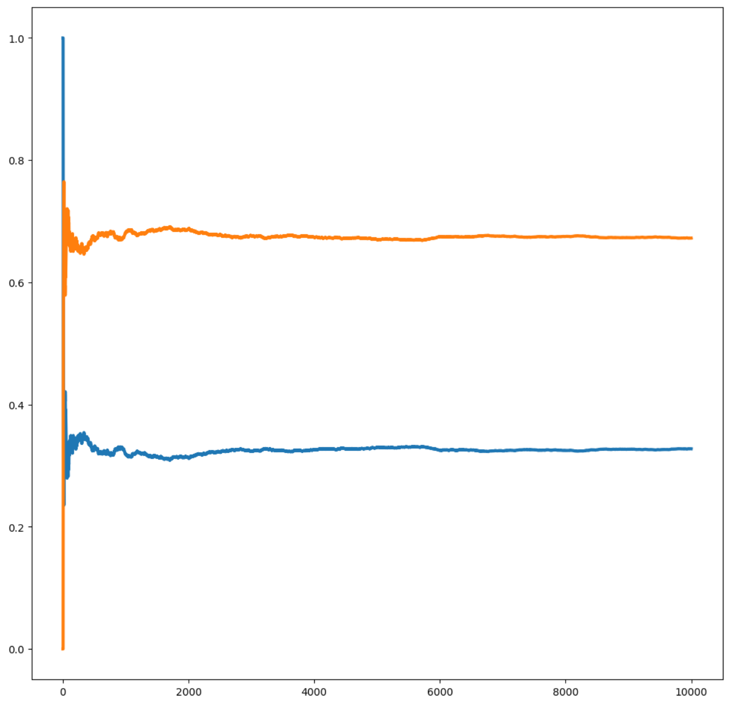
In [6]:
```
def mote_carlo_sim(n_sims):
    doors = [1,2,2] #1=car, 2=goat
    stick_wins = 0
    switch_wins = 0
    stick_wins_prob = np.zeros(n_sims)
    switch_wins_prob= np.zeros(n_sims)

    for i in range(n_sims):
        rand_dors = np.random.permutation(doors)
        choice = np.random.randint(3)

        if doors[choice] == 1:
            stick_wins = stick_wins + 1
        else:
            switch_wins = switch_wins + 1

        stick_wins_prob[i]=stick_wins/(i+1)
        switch_wins_prob[i]=switch_wins/(i+1)

    print(stick_wins_prob)
    print(switch_wins_prob)
    return [stick_wins, switch_wins, stick_wins_prob, switch_wins_prob]
```

In [7]:
```
n_sims=10000
[stick_wins, switch_wins, stick_wins_prob, switch_wins_prob] = mote_carlo_sim(n_sims)
print(stick_wins)
print(switch_wins)
```

```
[1.         1.         1.         ... 0.32766553 0.32763276 0.3276    ]
[0.         0.         0.         ... 0.67233447 0.67236724 0.6724    ]
3276
6724
```

In [8]:
```
plt.plot(np.arange(0,n_sims), stick_wins_prob, switch_wins_prob,linewidth=3) #plavo je stick_wins_prob
```

Out[8]:
```
[<matplotlib.lines.Line2D at 0x1e578afecc0>,
 <matplotlib.lines.Line2D at 0x1e578afeda0>]
```



Iz rezultata i sa grafika možemo videti da nakon 10,000 simulacija imamo skoro duplo veći broj slučajeva da je učesnik osvojio auto kada je zamenio vrata. Na taj način smo dobili i emprijsku potvrdu naše pretpostavke.

Monte Karlo integracija

Vraćamo se na današnju temu. Pokazaćemo na koji način koristimo Monte Karlo simulaciju da bi dobili vrednost određenog integrala.

Koristimo Teoremu Srednje Vrednosti koja je oblika:

$$f_{avg} = \frac{1}{b-a}\int_a^b f(x)dx$$

gde je f_{avg} prosečna vrednost funkcije na intervalu $[a, b]$.

Iz Teoreme Srednje Vrednosti sledi:

$$\int_a^b f(x)dx = (b-a)f_{avg}$$

Iz prethodne formule vidi se da ćemo određeni integrali funkcije na intervalu $[a, b]$ izračunavati pomoću proseka funkcije f_{avg} na tom intervalu. Postavlja se pitanja na koji način ćemo izračunati prosek?

Za izračunavanje proseka koristimo Monte Karlo princip, odnosno prosek procenjujemo pomoću uzorka (sempla) tačaka na intervalu $[a, b]$.

Sada možemo da kažemo da je Monte Karlo integracija stohastička metoda jer će rezultat, tj. prosek zavisiti od uzorka tačaka, a uzorak tačaka može biti svaki put drugačiji.

Naravno, što je uzorak veći to je procena stvarne vrednosti proseka bolja.

Formula za naivnu Monte Karlo metodu

U nastavku je data formula za naivnu Monte Karlo integraciju. Nakon naivne metode, predložene su mnoge varijante koje se uglavnom razlikuju po načinu na koji se uzrokuju tačke na osnovu kojih se procenjuje prosek.

Integral funkcije $f(x)$ na intervalu $[a, b]$ pomoću Monte Karlo integracije određujemo na sledeći način:

$$\int_a^b f(x)dx = (b - a)f_{avg}$$

$$f_{avg} \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i)$$

Tačke x_i, $i = 1, 2, \ldots, N$ su nasumično odbrane tačke iz unifromne distribucije na $[a, b]$, a N je veličina uzorka.

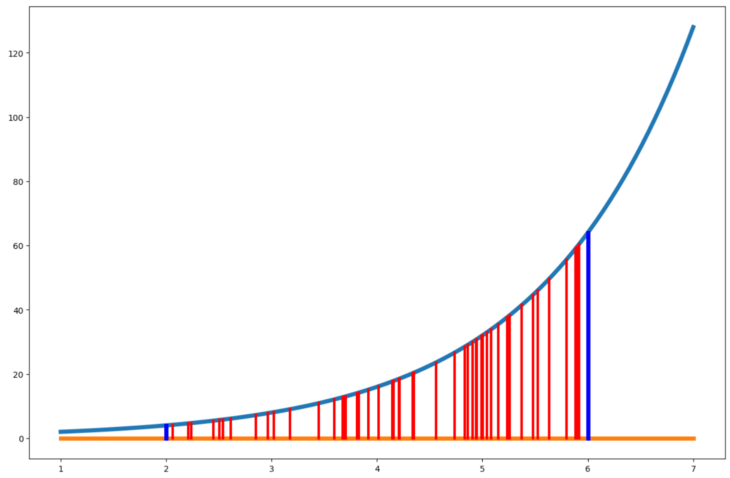
Pokazaćemo sada na primeru kako izgleda Monte Karlo integracija funkcije jedne promenljive. Primer koji koristimo je:

$$I = \int_2^6 2^x dx$$

Za početak pokazujemo kako izgleda jedan uzorak tačaka na intervalu $[2, 6]$.

In [9]:

```
a=2
b=6
N=50

plot_function([1,7],lambda x:2**x)
draw_vertical_lines([2,2**2],[6,2**6])

plot_sample(a,b,lambda x:2**x,N)
```



Na grafiku iznad prikazane su vrednosti funkcije $f(x) = 2^x$ u N tačaka koje su uzorkovane iz uniformne distribucije na intervalu $[2, 6]$.

Vrednosti određenog integrala predstavlja prosek tih N vrednosti funkcije $f(x) = 2^x$ pomnožen sa dužinom itervala, tj. $(6 - 2)$.

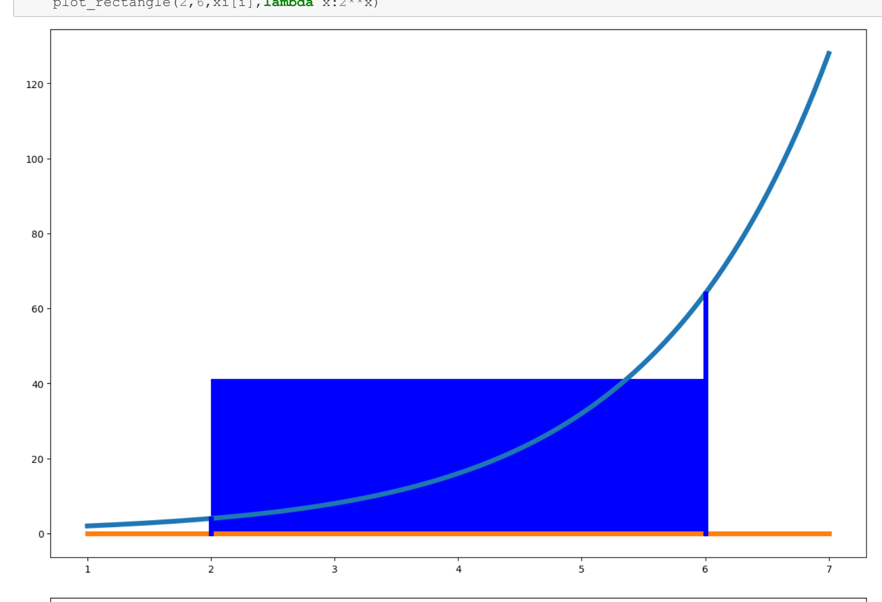
Ako je x_i uzorkovana tačka na intervalu $[a, b]$, onda je vrednost $f(x_i)(b - a)$ površina pravougaonika koji ima dužine stranica $(b - a)$ i $f(x_i)$. U nastavku pokazujemo 5 takvih pravougaonika za funkciju $f(x) = 2^x$ na intervalu $[2, 6]$.

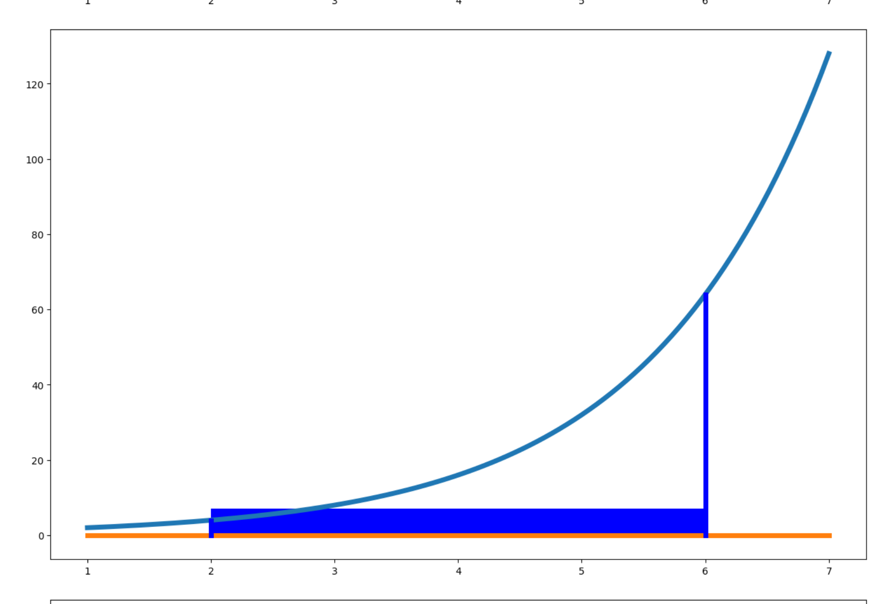
In [10]:

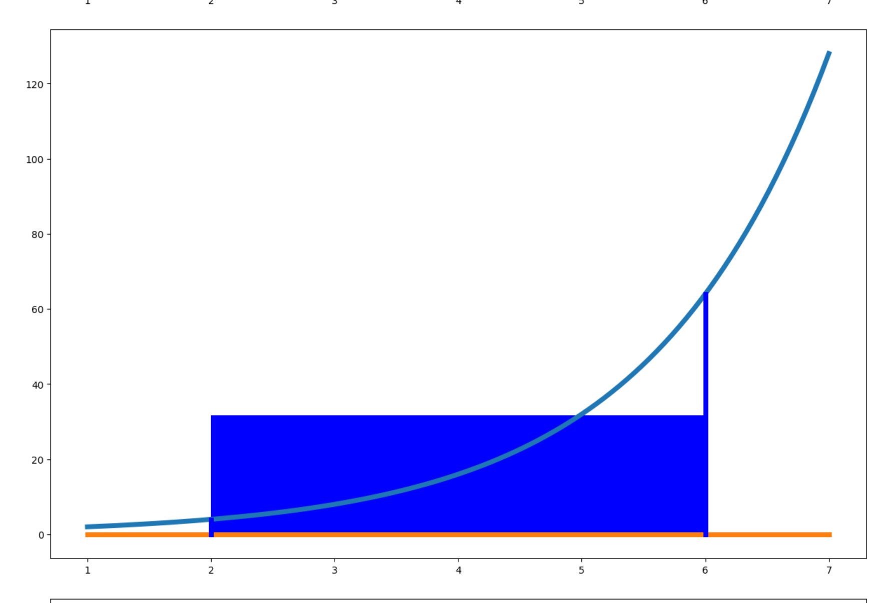
```
a=2
b=6
N=5
xi = np.random.rand(N)*(b-a)+a

print(xi)
```

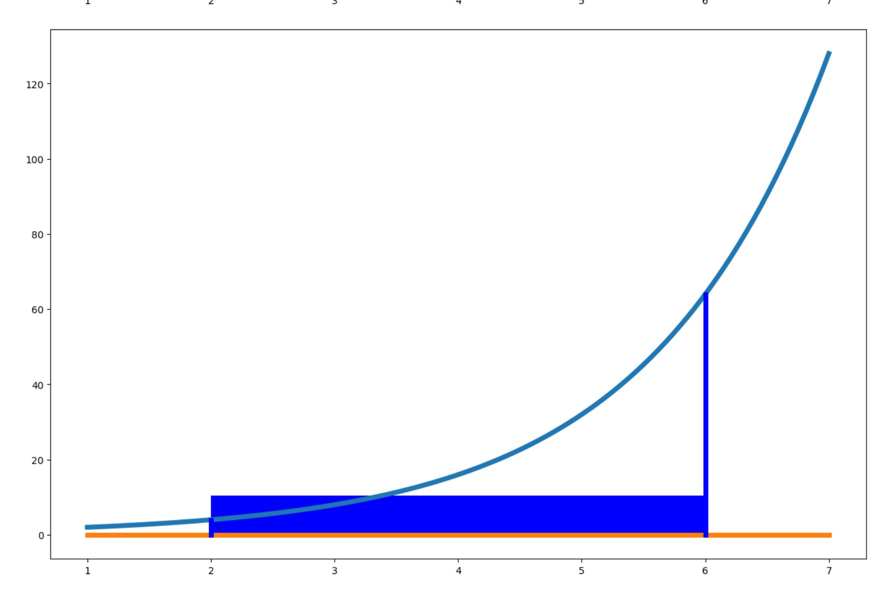
```
[5.36268513 2.83163347 4.98279743 3.3911474  3.19690446]
```

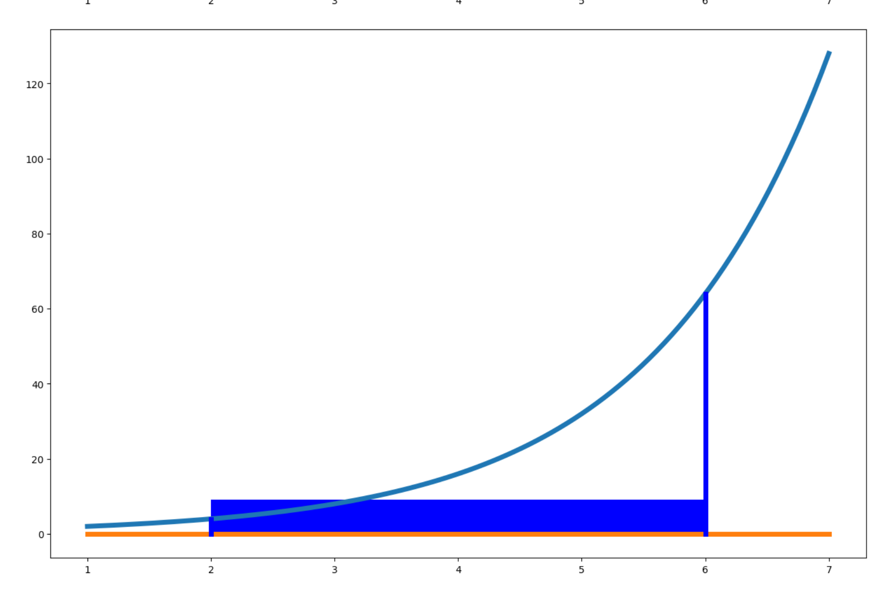
In [11]:
```python
for i in range(len(xi)):
    plot_function([1,7],lambda x:2**x)
    draw_vertical_lines([2,2**2],[6,2**6])
    plot_rectangle(2,6,xi[i],lambda x:2**x)
```











Procena površine ispod podintegralne funkcije u ovom slučaju bio bih prosek površina prikazanih 5 pravougaonika. Očigledno je da je 5 mali broj za dobru procenu, ali ako bi uzeli npr. 10,000 takvih pravougaonika dobili di prilično dobru procenu vrednosti određenog integrala.

Izračunaćemo sada vrednosti integrala

$$I = \int_2^6 2^x dx$$

pomoću 10,000 tačaka uzorkovanih iz uniformne distribucije na $[2, 6]$. Vrednost integrala pomoću Monte Karlo metode je:

$$\int_2^6 2^x dx = (6-2) f_{avg}$$

$$f_{avg} \approx \frac{1}{10,000} \sum_{i=1}^{10,000} 2^{x_i}$$

Vrednost integrala računamo pomoću Pythona.

In [12]:
```python
fun=lambda x:2**x
a=2
b=6
N=10000
xi=np.random.rand(N)*(b-a)+a
f_avg=np.mean(fun(xi))
I=(b-a)*f_avg

print(f_avg)
print(I)
```

```
21.697936331804737
86.79174532721895
```

In [13]:
```python
tacno_resenje = 86.56170
```

In [14]:
```python
procentualna_greska = abs(I-tacno_resenje)/tacno_resenje*100

print(procentualna_greska)
```

```
0.26575879080349135
```

Ako više puta pokrenemo kod za izračunavanje integrala, možemo da primetimo da se rezultat menja jer je metoda stohastička. Takođe se menja i greška koja je za ovaj primer pomoću 10,000 tačaka greška dosta mala (obično ispod 1%).

Rešićemo sada i primer dostrukog integrala koji smo rešavali na početku predavanja:

$$I = \int_0^{\frac{1}{2}} \int_0^{\frac{1}{2}} e^{y-x} dy dx$$

In [15]:
```python
fun=lambda x,y:np.exp(y-x)
a=0
b=0.5
c=0
d=0.5
N=10000

xi=np.random.rand(N)*(b-a)+a
yi=np.random.rand(N)*(d-c)+c

f_avg=np.mean(fun(xi,yi))

p_pravougaonik=(b-a)*(d-c)

I=p_pravougaonik*f_avg

print(f_avg)
print(I)
```

```
1.0228479216589343
0.2557119804147336
```

In [16]:
```python
tacno_resenje = 0.25525
```

In [17]:
```python
procentualna_greska = abs(I-tacno_resenje)/tacno_resenje*100

print(procentualna_greska)
```

```
0.18099134759397034
```

Dokaz da Monte Karlo integracija radi

[illegible]

$$\lim_{N \to \infty} f_{avg_N} = f_{avg}$$

kako povećavamo veličinu uzorka približavamo se tačnoj vednosti proseka funkcije f_{avg} na nekom intervalu. Time imamo dokaz da Monte Karlo integracija funkcioniše.

Greška Monte Karlo integracije

Cilj nam je da vidimo koliko tipično naša procena proseka funkcije pomoću uzorka f_{avg_N} odstupa od stvarne vrednosti proseka funkcije f_{avg}. To možemo da uradimo pomoću standardne devijacije. Dakle, hoćemo da vidimo koja je standardna devijacija naših procena proseka pomoću uzorka (označićemo je sa $\sigma_{f_{avg_N}}$) od stvarne vrednosti proseka funkcije. Takođe hoćemo da vidimo u kom je odnosu ta standardna devijacija sa veličinom uzroka N.

Odogovor na prethodna dva pitanja daje nam Centralna granična teorema koja ima sledeće intuitivno tumačenje (formalna definicija i dokaz su van opsega ovog predmeta i pripadaju oblasti verovatnoće i statistike):

Recimo da imamo neku distribuciju koja ima konačnu srednju vrednost μ i standardnu devijaciju σ koje nam nisu poznate.

Ako koristimo uzorke da procenimo μ i σ onda će rezultati tih procena biti normalno distribuirani (bez obzira iz kakve distribucije uzorkujemo).

Srednja vrednost te normalne distribucije je baš srednja vrednost distribucije iz koje uzorkujemo μ, a standardna devijacija zavisi od veličine uzorka N.

Što više tačaka uzorkujemo standardna devijacija je manja (normalna distribucija je "uža").

Kokretno vrednost standardne devijacije procena σ_P je:

$$\sigma_P = \frac{\sigma}{\sqrt{N}}$$

Šta možemo da zaključimo iz Centralne granične teoreme vezano za Monte Karlo integraciju:

1. Naše procena proseka pomoću uzorka su normalno distribuirane oko od stvarne vrednosti proseka funkcije.

2. Standardna devijacija procena proseka funkcije pomoću uzorka (označićemo je sa $\sigma_{f_{avg_N}}$) oko stvarne vrednosti proseka funkcije f_{avg} ima vrednost:

$$\sigma_{f_{avg_N}} = \frac{\sigma}{\sqrt{N}}$$

Dakle, tačka 1. nam kaže da ćemo uvek odbiti normalnu distribuciju procena srednje vrednosti, a tačka 2. da će se normalna distribucija sužavati oko stvarne srednje vrednosti kako povećavamo veličinu uzorka.

Šta još možemo da zaključimo iz 1. i 2.?

Prvo, poznato je (a to ćete detaljnije učiti na drugim predmetima) da se većina tačaka normalne distribucije nalazi untar 2 standardne devijacije oko srednje vrednosti, pa onda na osnovu 1. i 2. možemo da zaključimo da će rezultat Monte Karlo metode biti negde u sledećim granicma:

$$I = \int_a^b f(x)dx = (b-a) f_{avg} \pm \frac{\sigma}{\sqrt{N}}$$

gde je σ standardna devijacija funkcije $f(x)$ na intervalu $[a, b]$.

Drugo, ako povećavamo N standardna devijacija naših procena proseka prbližavaće se stvarnom proseku proprocionalno sa $\frac{1}{\sqrt{N}}$ što znači da je red greške Monte Karlo integracije:

$$O\left(\frac{1}{\sqrt{N}}\right)$$

Ako broj tačaka povećmo 4 puta imamo sledeće:

$$O\left(\frac{1}{\sqrt{4N}}\right) = \frac{1}{\sqrt{4}} O\left(\frac{1}{\sqrt{N}}\right) = \frac{1}{2} O\left(\frac{1}{\sqrt{N}}\right)$$

Znači, ako hoćemo da grešku smanjimo 2 puta, moramo 4 puta da povećamo broj tačaka.

Šta možemo da zaključimo iz reda greške Monte Karlo metode?

Podsetimo se reda grešaka Njutn-Kotesovih metoda:

Ako uzmemo da je $h = \frac{b-a}{N}$ greška metoda trapeza je:

$$O\left(h^2\right) = O\left(\frac{1}{N^2}\right)$$

Ako broj tačaka povećmo 4 puta imamo sledeće:

$$O\left(\frac{1}{(4N)^2}\right) = \frac{1}{16} O\left(\frac{1}{N^2}\right)$$

Očigledno je da već metoda trapeza ima mnogo bolji red greške od Monte Karlo metode. Postavlja se pitanje zašto bi koristili Monte Karlo metodu?

Zato što veličina uzorka N i samim tim i red greške ne zavise od dimezionalnost, tj. od višestrukosti integala.

Ako kod metode trapeza dupliramo broj tačaka, to dupliranje je po dimenziji. Na primer, ako imamo dvostruki integral i umesto 3 tačke po dimenziji uzmemo 6 tačaka onda sa ukupno 9 2d tačka prelazimo na 36 2d tačaka.

Dakle, broj tačaka koji nam treba da smanjimo grešku kod Njutn-kotesovih metoda zavisi od višestrukosti integrala. To nije slučaj kod Monte Karlo metode. Bez obzira da li radimo u 1d, 2d ili n-d ako broj tačaka povećamo 4 puta greška će se uvek prepoloviti.

Pokazaćemo sada na primeru sledećeg integala kako izgleda greška Monte Karlo metode:

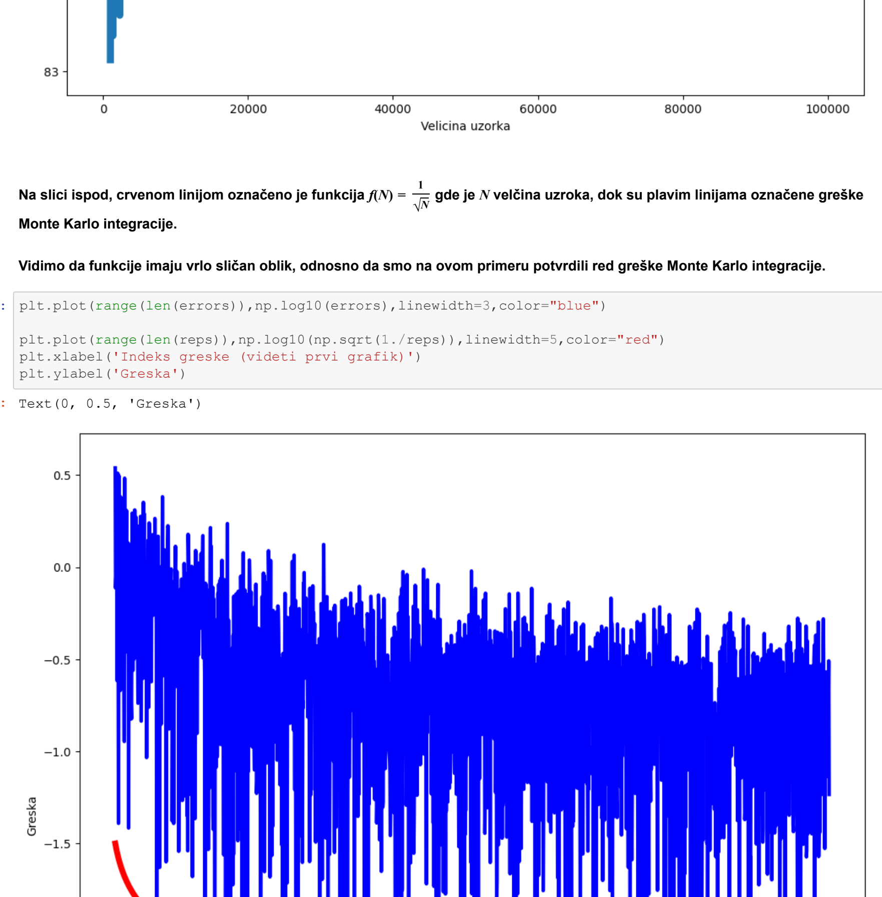
$$I = \int_2^6 2^x dx$$

Na slici ispod, crvenom linijom označeno je tačno rešenje, dok su plavim linijama označeni rezultati Monte Karlo integracije.

Vidimo da se sa povećanjem veličine uzroka rezultati sve manje variraju oko tačnog rešenja.

In [18]:
```python
fun=lambda x:2**x
a=2
b=6
max_N=100000
#max_N=1100
correct_solution=86.56170

[errors,reps]=calculate_error(a,b,fun,max_N,correct_solution)

print(errors)
```

```
[3.4530291  0.76924577 0.88346521 ... 0.18258788 0.30989586 0.05832923]
```

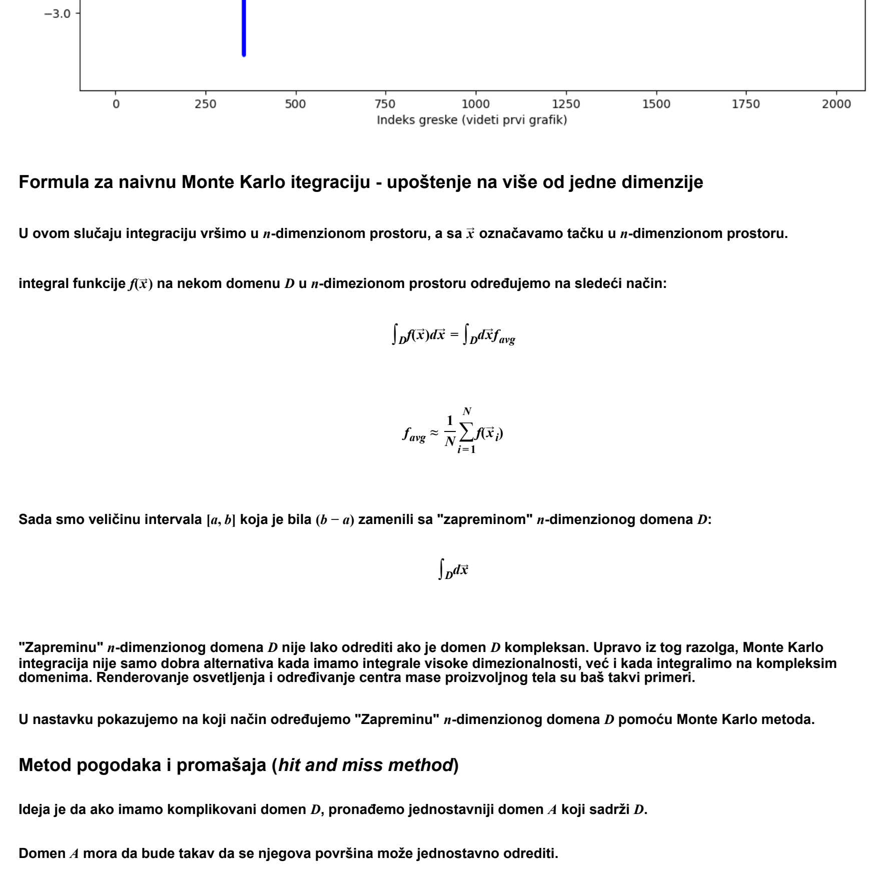


Na slici ispod, crvenom linijom označeno je funkcija $f(N) = \frac{1}{\sqrt{N}}$ gde je N velčina uzroka, dok su plavim linijama označene greške Monte Karlo integracije.

Vidimo da funkcije imaju vrlo sličan oblik, odnosno da smo na ovom primeru potvrdili red greške Monte Karlo integracije.

In [19]:
```python
plt.plot(range(len(errors)),np.log10(errors),linewidth=3,color="blue")

plt.plot(range(len(reps)),np.log10(np.sqrt(1./reps)),linewidth=5,color="red")
plt.xlabel('Indeks greske (videti prvi grafik)')
plt.ylabel('Greska')
```

Out[19]:
```
Text(0, 0.5, 'Greska')
```



Formula za naivnu Monte Karlo itegraciju - upoštenje na više od jedne dimenzije

U ovom slučaju integraciju vršimo u n-dimenzionom prostoru, a sa $\vec{x}$ označavamo tačku u n-dimenzionom prostoru.

integral funkcije $f(\vec{x})$ na nekom domenu D u n-dimezionom prostoru određujemo na sledeći način:

$$\int_D f(\vec{x}) d\vec{x} = \int_D d\vec{x} f_{avg}$$

$$f_{avg} \approx \frac{1}{N} \sum_{i=1}^{N} f(\vec{x}_i)$$

Sada smo veličinu intervala $[a, b]$ koja je bila $(b-a)$ zamenili sa "zapreminom" n-dimenzionog domena D:

$$\int_D d\vec{x}$$

"Zapreminu" n-dimenzionog domena D nije lako odrediti ako je domen D kompleksan. Upravo iz tog razolga, Monte Karlo integracija nije samo dobra alternativa kada imamo integrale visoke dimezionalnosti, već i kada integralimo na kompleksim domenima. Renderovanje osvetljenja i određivanje centra mase proizvoljnog tela su baš takvi primeri.

U nastavku pokazujemo na koji način određujemo "Zapreminu" n-dimenzionog domena D pomoću Monte Karlo metoda.

Metod pogodaka i promašaja (*hit and miss method*)

Ideja je da ako imamo komplikovani domen D, pronađemo jednostavniji domen A koji sadrži D.

Domen A mora da bude takav da se njegova površina može jednostavno odrediti.

Uzorkujemo N tačaka iz domena A.

Brojimo koliko od tih tačaka je u D i tu vrednost označimo sa N_D.

Površina domena D je onda:

$$P_D = \frac{N_D}{N} P_A$$

gde je P_A površina domena A.

Metod pogodaka i promašaja ilustrovaćemo na određivanju vrednosti broja π.

Kao domen A uzimamo jedinični kvadrat (označen crveno na slici dole), a kao domen D jedinični krug (označen plavo na slici dole).

Generišemo $N = 10,000$ tačaka iz jediničnog kvadrata i provervamo koji deo je u jediničnom krugu.

Površina kruga je $P_C = r^2 pi$, dok je površina kvadrata $P_S = širina^2 = (2r)^2$. Za $r = 1$ imamo $P_S = širina^2 = 4$. Onda važi da je vrednosti pi:

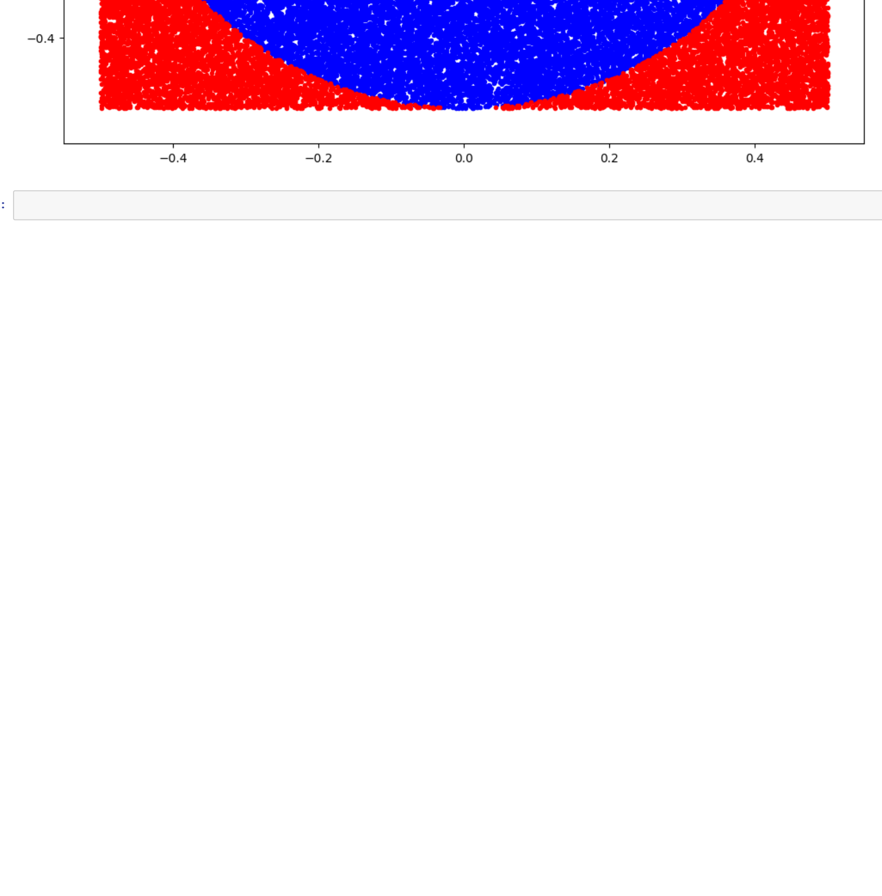
$$pi = P_C = \frac{N_C}{N_S} 4$$

gde je N_C broj slučajno generisanih tačaka koji pripada krugu, a N_S kvadratu.

U nastavku je dat Python kod.

In [20]:
```python
nmax = 100000 #velicina uzorka - broj slucajno generisanih tacaka

x = np.random.rand(nmax) #uzokovanje u izmedju 0 i 1
y = np.random.rand(nmax)

x1 = x - 0.5 #pomocu uzorka iz 0 i 1 dobijamo uzorak koji je u jedinicnom kvadratu
y1 = y - 0.5 #vrednosti x1 i y1 su u rasponu [-0.5,0.5]

r = np.sqrt(x1**2 + y1**2) #ubacjemo uzorak (2d tacku) u jedancinu kruga da dobijemo r
                           #ovo je ekulidsko rastojanje 2d tacke od koordinatnog pocetka (0,0)

inside = r <= 0.5 #tesitramo koji uzorci su u krugu, euklidsko rastojanje <= 0.5
outside = r > 0.5 #a koji nisu

plt.plot(x1[inside],y1[inside],'b.')

plt.plot(x1[outside],y1[outside],'r.')

thepi = 4 * sum(inside) / nmax #odredjujemo pi

print(thepi)
```

```
3.14004
```



In [ ]: