Generating Normal and Exponential Variates in Python, R, and C++

Thomas Walker

September 2022

1 Introduction

This is a secondary report to the primary report, where I will discuss how we can generate variates from other distributions across the three languages. We will mainly concern ourselves with the normal distribution and the exponential distribution.

2 Generating Exponential Variates

2.1 Python

For this python uses the inverse-CDF method by default, despite having an implementation of the Ziggurat method within the source code and stating within the documentation that the Ziggurat method is used.

2.2 R

R uses an algorithm attributed to JH Ahrens and U Dieter in order to generate the exponential variates.

2.3 C++

The standard library in C++ has an `exponential_distribution` function to generate exponential variates. It uses the inverse-CDF method in order to generate its variates. The boost library has a similar `exponential_distribution` function to generate variates from the standard exponential. However, it generates the variates using the Ziggurat method (explain in the appendix).

2.4 Code Implementation

```
def python_random_exponential_inv(state, size=1):
    output=[]
    def python_random_exponential_variate_inv(state):
        u1, state = python_random_uniform(state)
        return -np.log(1-u1[0]), state
    for n in range(size):
        e1, state = python_random_exponential_variate_inv(state)
        output.append(e1)
```

```
    return output, state

def r_random_exponential(state, size=1):
    output = []
    def r_random_exponential_variate(state):
        q = [
            0.6931471805599453,
            0.9333736875190459,
            0.9888777961838675,
            0.9984959252914960,
            0.9998292811061389,
            0.9999833164100727,
            0.9999985691438767,
            0.9999998906925558,
            0.9999999924734159,
            0.9999999995283275,
            0.9999999999728814,
            0.9999999999985598,
            0.9999999999999289,
            0.9999999999999968,
            0.9999999999999999,
            1.0000000000000000
            ]
        a = 0
        u, state = r_random_uniform(state)
        u1=u[0]
        while u1<=0 or u1>=1:
            u, state = r_random_uniform(state)
            u1=u[0]
        while True:
            u1+=u1
            if u1>1:
                break
            a+=q[0]
        u1-=1
        if u1<=q[0]:
            return a+u1, state
        i=0
        ustar, state = r_random_uniform(state)
        ustar1=ustar[0]
        umin=ustar1
        while u1>q[i]:
            ustar, state = r_random_uniform(state)
            ustar1=ustar[0]
            if umin>ustar1:
                umin=ustar1
            i+=1
```

```
        return a+umin*q[0], state
    for n in range(size):
        e1, state = r_random_exponential_variate(state)
        output.append(e1)
    return output, state

def cplusplus_random_exponential(state, size=1):
    output=[]
    def cplusplus_random_exponential_variate_inv(state):
        u1, state = cplusplus_boost_uniform_real_distribution(state)
        return -np.log(1-u1[0]), state
    for n in range(size):
        e1, state = cplusplus_random_exponential_variate_inv(state)
        output.append(e1)
    return output, state

def cplusplus_boost_exponential(state, size=1):
    table_x = [
        8.6971174701310497140, 7.6971174701310497140, 6.9410336293772123602, 6.
↪4783784938325698538,
        6.1441646657724730491, 5.8821443157953997963, 5.6664101674540337371, 5.
↪4828906275260628694,
        5.3230905057543986131, 5.1814872813015010392, 5.0542884899813047117, 4.
↪9387770859012514838,
        4.8329397410251125881, 4.7352429966017412526, 4.6444918854200854873, 4.
↪5597370617073515513,
        4.4802117465284221949, 4.4052876934735729805, 4.3344436803172730116, 4.
↪2672424802773661873,
        4.2033137137351843802, 4.1423408656640511251, 4.0840513104082974638, 4.
↪0282085446479365106,
        3.9746060666737884793, 3.9230625001354895926, 3.8734176703995089983, 3.
↪8255294185223367372,
        3.7792709924116678992, 3.7345288940397975350, 3.6912010902374189454, 3.
↪6491955157608538478,
        3.6084288131289096339, 3.5688252656483374051, 3.5303158891293438633, 3.
↪4928376547740601814,
        3.4563328211327607625, 3.4207483572511205323, 3.3860354424603017887, 3.
↪3521490309001100106,
        3.3190474709707487166, 3.2866921715990692095, 3.2550473085704501813, 3.
↪2240795652862645207,
        3.1937579032122407483, 3.1640533580259734580, 3.1349388580844407393, 3.
↪1063890623398246660,
        3.0783802152540905188, 3.0508900166154554479, 3.0238975044556767713, 2.
↪9973829495161306949,
        2.9713277599210896472, 2.9457143948950456386, 2.9205262865127406647, 2.
↪8957477686001416838,
```

```
       2.8713640120155362592, 2.8473609656351888266, 2.8237253024500354905, 2.
↪8004443702507381944,
       2.7775061464397572041, 2.7548991965623453650, 2.7326126361947007411, 2.
↪7106360958679293686,
       2.6889596887418041593, 2.6675739807732670816, 2.6464699631518093905, 2.
↪6256390267977886123,
       2.6050729387408355373, 2.5847638202141406911, 2.5647041263169053687, 2.
↪5448866271118700928,
       2.5253043900378279427, 2.5059507635285939648, 2.4868193617402096807, 2.
↪4679040502973649846,
       2.4491989329782498908, 2.4306983392644199088, 2.4123968126888708336, 2.
↪3942890999214583288,
       2.3763701405361408194, 2.3586350574093374601, 2.3410791477030346875, 2.
↪3236978743901964559,
       2.3064868582835798692, 2.2894418705322694265, 2.2725588255531546952, 2.
↪2558337743672190441,
       2.2392628983129087111, 2.2228425031110364013, 2.2065690132576635755, 2.
↪1904389667232199235,
       2.1744490099377744673, 2.1585958930438856781, 2.1428764653998416425, 2.
↪1272876713173679737,
       2.1118265460190418108, 2.0964902118017147637, 2.0812758743932248696, 2.
↪0661808194905755036,
       2.0512024094685848641, 2.0363380802487695916, 2.0215853383189260770, 2.
↪0069417578945183144,
       1.9924049782135764992, 1.9779727009573602295, 1.9636426877895480401, 1.
↪9494127580071845659,
       1.9352807862970511135, 1.9212447005915276767, 1.9073024800183871196, 1.
↪8934521529393077332,
       1.8796917950722108462, 1.8660195276928275962, 1.8524335159111751661, 1.
↪8389319670188793980,
       1.8255131289035192212, 1.8121752885263901413, 1.7989167704602903934, 1.
↪7857359354841254047,
       1.7726311792313049959, 1.7596009308890742369, 1.7466436519460739352, 1.
↪7337578349855711926,
       1.7209420025219350428, 1.7081947058780575683, 1.6955145241015377061, 1.
↪6829000629175537544,
       1.6703499537164519163, 1.6578628525741725325, 1.6454374393037234057, 1.
↪6330724165359912048,
       1.6207665088282577216, 1.6085184617988580769, 1.5963270412864831349, 1.
↪5841910325326886695,
       1.5721092393862294810, 1.5600804835278879161, 1.5481036037145133070, 1.
↪5361774550410318943,
       1.5243009082192260050, 1.5124728488721167573, 1.5006921768428164936, 1.
↪4889578055167456003,
```

```
        1.4772686611561334579, 1.4656236822457450411, 1.4540218188487932264, 1.4424620319720121876,
        1.4309432929388794104, 1.4194645827699828254, 1.4080248915695353509, 1.3966232179170417110,
        1.3852585682631217189, 1.3739299563284902176, 1.3626364025050864742, 1.3513769332583349176,
        1.3401505805295045843, 1.3289563811371163220, 1.3177933761763245480, 1.3066606104151739482,
        1.2955571316866007210, 1.2844819902750125450, 1.2734342382962410994, 1.2624129290696153434,
        1.2514171164808525098, 1.2404458543344064544, 1.2294981956938491599, 1.2185731922087903071,
        1.2076698934267612830, 1.1967873460884031665, 1.1859245934042023557, 1.1750806743109117687,
        1.1642546227056790397, 1.1534454666557748056, 1.1426522275816728928, 1.1318739194110786733,
        1.1211095477013306083, 1.1103581087274114281, 1.0996185885325976575, 1.0888899619385472598,
        1.0781711915113727024, 1.0674612264799681530, 1.0567590016025518414, 1.0460634359770445503,
        1.0353734317905289496, 1.0246878730026178052, 1.0140056239570971074, 1.0033255279156973717,
        0.99264640550727647009, 0.98196705308506317914, 0.97128624098390397896, 0.96060271166866709917,
        0.94991517776407659940, 0.93922231995526297952, 0.92852278474721113999, 0.91781518207004493915,
        0.90709808271569100600, 0.89637001558989069006, 0.88562946476175228052, 0.87487486629102585352,
        0.86410460481100519511, 0.85331700984237406386, 0.84251035181036928333, 0.83168283773427388393,
        0.82083260655441252290, 0.80995772405741906620, 0.79905617735548788109, 0.78812586886949324977,
        0.77716460975913043936, 0.76617011273543541328, 0.75513998418198289808, 0.74407171550050873971,
        0.73296267358436604916, 0.72181009030875689912, 0.71061105090965570413, 0.69936248110323266174,
        0.68806113277374858613, 0.67670356802952337911, 0.66528614139267855405, 0.65380497984766565353,
        0.64225596042453703448, 0.63063468493349100113, 0.61893645139487678178, 0.60715622162030085137,
        0.59528858429150359384, 0.58332771274877027785, 0.57126731653258903915, 0.55910058551154127652,
        0.54682012516331112550, 0.53441788123716615385, 0.52188505159213564105, 0.50921198244365495319,
```

```
        0.49638804551867159754, 0.48340149165346224782, 0.47023927508216945338,␣
↪0.45688684093142071279,
        0.44332786607355296305, 0.42954394022541129589, 0.41551416960035700100,␣
↪0.40121467889627836229,
        0.38661797794112021568, 0.37169214532991786118, 0.35639976025839443721,␣
↪0.34069648106484979674,
        0.32452911701691008547, 0.30783295467493287307, 0.29052795549123115167,␣
↪0.27251318547846547924,
        0.25365836338591284433, 0.23379048305967553619, 0.21267151063096745264,␣
↪0.18995868962243277774,
        0.16512762256418831796, 0.13730498094001380420, 0.10483850756582017915,␣
↪0.06385216381500348017З,
        0
        ]
    table_y = [
        0, 0.00045413435384149675545, 0.00096726928232717452884, 0.
↪0015362997803015723824,
        0.0021459677437189061793, 0.0027887987935740759640, 0.
↪0034602647778369039855, 0.0041572951208337952532,
        0.0048776559835423925804, 0.0056196422072054831710, 0.
↪0063819059373191794422, 0.0071633531836349841425,
        0.0079630774380170392396, 0.0087803149858089752347, 0.
↪0096144136425022094101, 0.010464810181029979488,
        0.011331013597834597488, 0.012212592426255380661, 0.
↪013109164931254991070, 0.014020391403181937334,
        0.014945968011691148079, 0.015885621839973162490, 0.
↪016839106826039946359, 0.017806200410911360563,
        0.018786700744696029497, 0.019780424338009741737, 0.
↪020787204072578117603, 0.021806887504283582125,
        0.022839335406385238829, 0.023884420511558170348, 0.
↪024942026419731782971, 0.026012046645134218076,
        0.027094383780955798424, 0.028188948763978634421, 0.
↪029295660224637394015, 0.030414443910466605492,
        0.031545232172893605499, 0.032687963508959533317, 0.
↪033842582150874329031, 0.035009037697397411067,
        0.036187284781931419754, 0.037377282772959360128, 0.
↪038578995503074859626, 0.039792391023374122670,
        0.041017441380414820816, 0.042254122413316231413, 0.
↪043502413568888183301, 0.044762297732943280694,
        0.046033761076175166762, 0.047316792913181548703, 0.
↪048611385573379494401, 0.049917534282706374944,
        0.051235237055126279830, 0.052564494593071689595, 0.
↪053905310196046085104, 0.055257689676697038322,
        0.056621641283742874438, 0.057997175631200659098, 0.
↪059384305633420264487, 0.060783046445479636051,
```

```
    0.062193415408540996150, 0.063615431999807331076, 0.
↪065049117786753755036, 0.066494496385339779043,
    0.067951593421936607770, 0.069420436498728751675, 0.
↪070901055162371828426, 0.072393480875708743023,
    0.073897746992364746308, 0.075413888734058408453, 0.
↪076941943170480510100, 0.078481949201606426042,
    0.080033947542319910023, 0.081597980709237420930, 0.
↪083174093009632380354, 0.084762330532368125386,
    0.086362741140756912277, 0.087975374467270219300, 0.
↪089600281910032864534, 0.091237516631040162057,
    0.092887133556043546523, 0.094549189376055853718, 0.
↪096223742550432800103, 0.097910853311492199618,
    0.099610583670637128826, 0.10132299742595363588, 0.10304816017125771553,␣
↪0.10478613930657016928,
    0.10653700405000166218, 0.10830082545103379867, 0.11007767640518539026,␣
↪0.11186763167005629731,
    0.11367076788274431301, 0.11548716357863353664, 0.11731689921155557057,␣
↪0.11916005717532768467,
    0.12101672182667483729, 0.12288697950954513498, 0.12477091858083096578,␣
↪0.12666862943751066518,
    0.12858020454522817870, 0.13050573846833078225, 0.13244532790138752023,␣
↪0.13439907170221363078,
    0.13636707092642885841, 0.13834942886358021406, 0.14034625107486244210,␣
↪0.14235764543247220043,
    0.14438372216063476473, 0.14642459387834493787, 0.14848037564386679222,␣
↪0.15055118500103990354,
    0.15263714202744286154, 0.15473836938446807312, 0.15685499236936522013,␣
↪0.15898713896931420572,
    0.16113493991759203183, 0.16329852875190180795, 0.16547804187493600915,␣
↪0.16767361861725019322,
    0.16988540130252766513, 0.17211353531532005700, 0.17435816917135348788,␣
↪0.17661945459049489581,
    0.17889754657247831241, 0.18119260347549629488, 0.18350478709776746150,␣
↪0.18583426276219711495,
    0.18818119940425430485, 0.19054576966319540013, 0.19292814997677133873,␣
↪0.19532852067956322315,
    0.19774706610509886464, 0.20018397469191127727, 0.20263943909370901930,␣
↪0.20511365629383770880,
    0.20760682772422204205, 0.21011915938898825914, 0.21265086199297827522,␣
↪0.21520215107537867786,
    0.21777324714870053264, 0.22036437584335949720, 0.22297576805812018050,␣
↪0.22560766011668406495,
    0.22826029393071670664, 0.23093391716962742173, 0.23362878343743333945,␣
↪0.23634515245705964715,
```

```
        0.23908329026244917002, 0.24184346939887722761, 0.24462596913189210901,␣
↪0.24743107566532763894,
        0.25025908236886230967, 0.25311029001562948171, 0.25598500703041538015,␣
↪0.25888354974901621678,
        0.26180624268936295243, 0.26475341883506220209, 0.26772541993204481808,␣
↪0.27072259679906003167,
        0.27374530965280298302, 0.27679392844851734458, 0.27986883323697289920,␣
↪0.28297041453878076010,
        0.28609907373707684673, 0.28925522348967773308, 0.29243928816189258772,␣
↪0.29565170428126120948,
        0.29889292101558177099, 0.30216340067569352897, 0.30546361924459023541,␣
↪0.30879406693456016794,
        0.31215524877417956945, 0.31554768522712893632, 0.31897191284495723773,␣
↪0.32242848495608914289,
        0.32591797239355619822, 0.32944096426413633091, 0.33299806876180896713,␣
↪0.33658991402867758144,
        0.34021714906678004560, 0.34388044470450243010, 0.34758049462163698567,␣
↪0.35131801643748334681,
        0.35509375286678745925, 0.35890847294874976196, 0.36276297335481777335,␣
↪0.36665807978151414890,
        0.37059464843514599421, 0.37457356761590215193, 0.37859575940958081092,␣
↪0.38266218149600982112,
        0.38677382908413768115, 0.39093173698479710717, 0.39513698183329015336,␣
↪0.39939068447523107877,
        0.40369401253053026739, 0.40804818315203238238, 0.41245446599716116772,␣
↪0.41691418643300289465,
        0.42142872899761659635, 0.42599954114303435739, 0.43062813728845883923,␣
↪0.43531610321563659758,
        0.44006510084235387501, 0.44487687341454851593, 0.44975325116275498919,␣
↪0.45469615747461548049,
        0.45970761564213768669, 0.46478975625042618067, 0.46994482528395999841,␣
↪0.47517519303737738299,
        0.48048336393045423016, 0.48587198734188493564, 0.49134386959403255500,␣
↪0.49690198724154955294,
        0.50254950184134769289, 0.50828977641064283495, 0.51412639381474855788,␣
↪0.52006317736823356823,
        0.52610421398361972602, 0.53225388026304326945, 0.53851687200286186590,␣
↪0.54489823767243963663,
        0.55140341654064131685, 0.55803828226258748140, 0.56480919291240022434,␣
↪0.57172304866482579008,
        0.57878735860284503057, 0.58601031847726802755, 0.59340090169173341521,␣
↪0.60096896636523224742,
        0.60872538207962206507, 0.61668218091520762326, 0.62485273870366592605,␣
↪0.63325199421436607968,
```

```
        0.64189671642726607018, 0.65080583341457104881, 0.66000084107899974178, 0.66950631673192477684,
        0.67935057226476538741, 0.68956649611707798890, 0.70019265508278816709, 0.71127476080507597882,
        0.72286765959357200702, 0.73503809243142351530, 0.74786862198519510742, 0.76146338884989624862,
        0.77595685204011559675, 0.79152763697249565519, 0.80842165152300838005, 0.82699329664305033399,
        0.84778550062398962096, 0.87170433238120363669, 0.90046992992574643800, 0.93814368086217467916,
        1
        ]
    output=[]
    def variate(state):
        shift=0
        while True:
            j, i, state = cplusplus_boost_generate_int_float_pair(state, 8)
            x = table_x[i]*j
            if x<table_x[i+1]:
                return shift+x, state
            if i==0:
                shift+=table_x[1]
            else:
                y_01, state = cplusplus_boost_uniform_01(state)
                y_01 = y_01[0]
                y = table_y[i]+y_01*(table_y[i+1]-table_y[i])
                y_above_ubound = (table_x[i]-table_x[i+1])*y_01 - (table_x[i]-x)
                y_above_lbound = y - table_y[i+1]+(table_x[i+1]-x)*table_y[i+1]
                if (y_above_ubound > 0 > y_above_lbound) or (y<np.exp(-x)):
                    return x+shift, state
    for n in range(size):
        e, state = variate(state)
        output.append(e)
    return output, state
```

2.5 Testing

```
print('Python')
print('Variates from numpy.standard_exponential:')
rng.seed(1)
print(rng.standard_exponential(size=5))
print('Variates using our function:')
e_p=python_random_exponential_inv(python_state_from_seed(1), size=5)[0]
print(e_p)
print('\nR')
print('Variates from rexp:')
```

```
robjects.r('''
set.seed(1)
print(rexp(5))
''')
print('Variates from our function:')
e_r= r_random_exponential(r_state_from_seed(1), size=5)[0]
print(e_r)
print('\nC++')
print('Variates from the standard random library:')
e_c1 = cplusplus_random_exponential(cplusplus_state_from_seed(1), size=5)[0]
print(e_c1)
print('Variates from the boost random library:')
e_c2 = cplusplus_boost_exponential(cplusplus_state_from_seed(1), size=5)[0]
print(e_c2)
```

```
Python
Variates from numpy.standard_exponential:
[5.39605837e-01 1.27412525e+00 1.14381359e-04 3.60012755e-01
 1.58709595e-01]
Variates using our function:
[0.5396058372591854, 1.2741252530133043, 0.00011438135864308592,
0.360012754853919, 0.1587095951946739]

R
Variates from rexp:
[1] 0.7551818 1.1816428 0.1457067 0.1397953 0.4360686
Variates from our function:
[0.7551818333756194, 1.1816427794536732, 0.14570672697054854,
0.13979526190104644, 0.43606862588070483]

C++
Variates from the standard random library:
[5.872725054900336, 2.6964778899049326, 0.13710857819070843, 6.949115421696289,
0.26930395910760874]
Variates from the boost random library:
[3.343707856277791, 0.7018779056655744, 0.001596022007909643,
0.2799580415572925, 0.7017106675875586]
```

3 Generating Normal Variates

3.1 Python

3.2 R

To generate its normal variates R uses the inverse-CDF method. However, by looking through the source code it is clear that it has the ability to use a vast array of other algorithms. The algorithms are implemented such that they sample from a standard normal, the variates are subsequently scaled if we are considering non-standard normal distributions.

3.3 C++

The boost library generates its variates using the Ziggurat method.

3.4 Code Implementation

```
def r_random_normal(state, mu=0, sigma=1, size=1):
    output = []
    def r_qnorm(p, mu, sigma, lower_tail, log_p):
        def R_D_Cval(p, lower_tail):
            if lower_tail:
                return 0.5-p+0.5
            else:
                return p
        def R_D_qIv(p, log_p):
            if log_p:
                return np.exp(p)
            else:
                return p
        def R_D_Lval(p, lower_tail):
            if lower_tail:
                return p
            else:
                return 0.5-p+0.5
        def R_DT_qIv(p, lower_tail, log_p):
            return R_D_Lval(R_D_qIv(p, log_p), lower_tail)
        p_ = R_DT_qIv(p, lower_tail, log_p)
        q = p_-0.5
        if abs(q)<=0.425:
            r=0.180625 - q**2
            val = q * (((((((r * 2509.0809287301226727 + 33430.575583588128105)␣
↪* r + 67265.770927008700853) * r + 45921.953931549871457) * r + 13731.
↪693765509461125) * r + 1971.5909503065514427) * r + 133.14166789178437745) * r␣
↪+ 3.387132872796366608) / (((((((r * 5226.495278852854561 + 28729.
↪085735721942674) * r + 39307.89580009271061) * r + 21213.794301586595867) * r␣
↪+ 5394.1960214247511077) * r + 687.1870074920579083) * r + 42.
↪313330701600911252) * r + 1.)
        else:
            if log_p and ((lower_tail and q <= 0) or (not(lower_tail) and q >␣
↪0)):
                r=p
            else:
                if q>0:
                    r=np.log(R_D_Cval(R_D_qIv(p,log_p), lower_tail))
                else:
                    r=np.log(p_)
            r=np.sqrt(-r)
```

```
            if r<=5:
                r+=(-1.6)
                val = (((((((r * 7.7454501427834140764e-4 + .
↪0227238449892691845833) * r + .24178072517745061177) * r + 1.
↪27045825245236838258) * r + 3.64784832476320460504) * r + 5.
↪7694972214606914055) * r + 4.6303378461565452959) * r + 1.
↪42343711074968357734) / (((((((r * 1.05075007164441684324e-9 + 5.
↪475938084995344946e-4) * r + .0151986665636164571966) * r + .
↪14810397642748007459) * r + .68976733498510000455) * r + 1.
↪6763848301838038494) * r + 2.05319162663775882187) * r + 1.)
            elif r>=816:
                val = r * 1.414213562373095048880
            else:
                r+= (-5)
                val = (((((((r * 2.01033439929228813265e-7 + 2.
↪71155556874348757815e-5) * r + .0012426609473880784386) * r + .
↪026532189526576123093) * r + .29656057182850489123) * r + 1.
↪7848265399172913358) * r + 5.4637849111641143699) * r + 6.6579046435011037772)␣
↪/ (((((((r * 2.04426310338993978564e-15 + 1.4215117583164458887e-7)* r + 1.
↪8463183175100546818e-5) * r + 7.868691311456132591e-4) * r + .
↪0148753612908506148525) * r + .13692988092273580531) * r + .
↪599832206555887937690) * r + 1.)
            if q<0:
                val = -val
        return mu + sigma * val
    def r_random_normal_variate(state):
        big = 134217728
        u1, state = r_random_uniform(state)
        u2, state = r_random_uniform(state)
        u = int(big*u1[0]) + u2[0]
        return mu+sigma*r_qnorm(u/big, mu, sigma, True, False), state
    for n in range(size):
        n1, state = r_random_normal_variate(state)
        output.append(n1)
    return output, state

def cplusplus_boost_normal(state, mu=0, sigma=1, size=1):
    table_x = [
        3.7130862467403632609, 3.4426198558966521214, 3.2230849845786185446, 3.
↪0832288582142137009,
        2.9786962526450169606, 2.8943440070186706210, 2.8231253505459664379, 2.
↪7611693723841538514,
        2.7061135731187223371, 2.6564064112581924999, 2.6109722484286132035, 2.
↪5690336259216391328,
        2.5300096723854666170, 2.4934545220919507609, 2.4590181774083500943, 2.
↪4264206455302115930,
```

```
        2.3954342780074673425, 2.3658713701139875435, 2.3375752413355307354, 2.
↪3104136836950021558,
        2.2842740596736568056, 2.2590595738653295251, 2.2346863955870569803, 2.
↪2110814088747278106,
        2.1881804320720206093, 2.1659267937448407377, 2.1442701823562613518, 2.
↪1231657086697899595,
        2.1025731351849988838, 2.0824562379877246441, 2.0627822745039633575, 2.
↪0435215366506694976,
        2.0246469733729338782, 2.0061338699589668403, 1.9879595741230607243, 1.
↪9701032608497132242,
        1.9525457295488889058, 1.9352692282919002011, 1.9182573008597320303, 1.
↪9014946531003176140,
        1.8849670357028692380, 1.8686611409895420085, 1.8525645117230870617, 1.
↪8366654602533840447,
        1.8209529965910050740, 1.8054167642140487420, 1.7900469825946189862, 1.
↪7748343955807692457,
        1.7597702248942318749, 1.7448461281083765085, 1.7300541605582435350, 1.
↪7153867407081165482,
        1.7008366185643009437, 1.6863968467734863258, 1.6720607540918522072, 1.
↪6578219209482075462,
        1.6436741568569826489, 1.6296114794646783962, 1.6156280950371329644, 1.
↪6017183802152770587,
        1.5878768648844007019, 1.5740982160167497219, 1.5603772223598406870, 1.
↪5467087798535034608,
        1.5330878776675560787, 1.5195095847593707806, 1.5059690368565502602, 1.
↪4924614237746154081,
        1.4789819769830978546, 1.4655259573357946276, 1.4520886428822164926, 1.
↪4386653166774613138,
        1.4252512545068615734, 1.4118417124397602509, 1.3984319141236063517, 1.
↪3850170377251486449,
        1.3715922024197322698, 1.3581524543224228739, 1.3446927517457130432, 1.
↪3312079496576765017,
        1.3176927832013429910, 1.3041418501204215390, 1.2905495919178731508, 1.
↪2769102735516997175,
        1.2632179614460282310, 1.2494664995643337480, 1.2356494832544811749, 1.
↪2217602305309625678,
        1.2077917504067576028, 1.1937367078237721994, 1.1795873846544607035, 1.
↪1653356361550469083,
        1.1509728421389760651, 1.1364898520030755352, 1.1218769225722540661, 1.
↪1071236475235353980,
        1.0922188768965537614, 1.0771506248819376573, 1.0619059636836193998, 1.
↪0464709007525802629,
        1.0308302360564555907, 1.0149673952392994716, 0.99886423348064351303, 0.
↪98250080350276038481,
```

```
        0.96585507938813059489, 0.94890262549791195381, 0.93161619660135381056, 0.91396525100880177644,
        0.89591535256623852894, 0.87742742909771569142, 0.85845684317805086354, 0.83895221428120745572,
        0.81885390668331772331, 0.79809206062627480454, 0.77658398787614838598, 0.75423066443451007146,
        0.73091191062188128150, 0.70647961131360803456, 0.68074791864590421664, 0.65347863871504238702,
        0.62435859730908822111, 0.59296294244197797913, 0.55869217837551797140, 0.52065603872514491759,
        0.47743783725378787681, 0.42654798630330512490, 0.36287143102841830424, 0.27232086470466385065,
        0
    ]

    table_y = [
        0, 0.0026696290839025035092, 0.0055489952208164705392, 0.0086244844129304709682,
        0.011839478657982313715, 0.015167298010672042468, 0.018592102737165812650, 0.022103304616111592615,
        0.025693291936149616572, 0.029356317440253829618, 0.033087886146505155566, 0.036884388786968774128,
        0.040742868074790604632, 0.044660862200872429800, 0.048636295860284051878, 0.052667401903503169793,
        0.056752663481538584188, 0.060890770348566375972, 0.065080585213631873753, 0.069321117394180252601,
        0.073611501884754893389, 0.077950982514654714188, 0.082338898242957408243, 0.086774671895542968998,
        0.091257800827634710201, 0.09578784912257815216, 0.10036444102954554013, 0.10498725541035453978,
        0.10965602101581776100, 0.11437051244988827452, 0.11913054670871858767, 0.12393598020398174246,
        0.12878670619710396109, 0.13368265258464764118, 0.13862377998585103702, 0.14361008009193299469,
        0.14864157424369696566, 0.15371831220958657066, 0.15884037114093507813, 0.16400785468492774791,
        0.16922089223892475176, 0.17447963833240232295, 0.17978427212496211424, 0.18513499701071343216,
        0.19053204032091372112, 0.19597565311811041399, 0.20146611007620324118, 0.20700370944187380064,
        0.21258877307373610060, 0.21822164655637059599, 0.22390269938713388747, 0.22963232523430270355,
        0.23541094226572765600, 0.24123899354775131610, 0.24711694751469673582, 0.25304529850976585934,
```

```
        0.25902456739871074263, 0.26505530225816194029, 0.27113807914102527343, 0.27727350292189771153,
        0.28346220822601251779, 0.28970486044581049771, 0.29600215684985583659, 0.30235482778947976274,
        0.30876363800925192282, 0.31522938806815752222, 0.32175291587920862031, 0.32833509837615239609,
        0.33497685331697116147, 0.34167914123501368412, 0.34844296754987246935, 0.35526938485154714435,
        0.36215949537303321162, 0.36911445366827513952, 0.37613546951445442947, 0.38322381105988364587,
        0.39038080824138948916, 0.39760785649804255208, 0.40490642081148835099, 0.41227804010702462062,
        0.41972433205403823467, 0.42724699830956239880, 0.43484783025466189638, 0.44252871528024661483,
        0.45029164368692696086, 0.45813871627287196483, 0.46607215269457097924, 0.47409430069824960453,
        0.48220764633483869062, 0.49041482528932163741, 0.49871863547658432422, 0.50712205108130458951,
        0.51562823824987205196, 0.52424057267899279809, 0.53296265938998758838, 0.54179835503172412311,
        0.55075179312105527738, 0.55982741271069481791, 0.56902999107472161225, 0.57836468112670231279,
        0.58783705444182052571, 0.59745315095181228217, 0.60721953663260488551, 0.61714337082656248870,
        0.62723248525781456578, 0.63749547734314487428, 0.64794182111855080873, 0.65858200005865368016,
        0.66942766735770616891, 0.68049184100641433355, 0.69178914344603585279, 0.70333609902581741633,
        0.71515150742047704368, 0.72725691835450587793, 0.73967724368333814856, 0.75244155918570380145,
        0.76558417390923599480, 0.77914608594170316563, 0.79317701178385921053, 0.80773829469612111340,
        0.82290721139526200050, 0.83878360531064722379, 0.85550060788506428418, 0.87324304892685358879,
        0.89228165080230272301, 0.91304364799203805999, 0.93628268170837107547, 0.96359969315576759960,
        1
    ]
    output=[]
    def generate_tail():
        tail_start = table_x[1]
        while True:
            x, state = cplusplus_boost_exponential(state)
            x = x[0]/tail_start
            y, state = cplusplus_boost_exponential(state)
            if 2*y[0]>x[0]**2:
```

```
                return x[0]+tail_start, state
    def variate(state):
        while True:
            j, i, state = cplusplus_boost_generate_int_float_pair(state, 8)
            sign = (i&1)*2-1
            i = i>>1
            x = j*table_x[i]
            if x<table_x[i+1]:
                return x*sign, state
            if i==0:
                return generate_tail()*sign, state
            y_01, state = cplusplus_boost_uniform_01(state)
            y_01=y_01[0]
            y = table_y[i]+y_01*(table_y[i+1]-table_y[i])
            if table_x[i]>=1:
                y_above_ubound = (table_x[i]-table_x[i+1])*y_01-(table_x[i]-x)
                y_above_ubound = y -␣
↪(table_y[i]+(table_x[i]-x)*table_y[i]*table_x[i])
            else:
                y_above_lbound = (table_x[i]-table_x[i+1])*y_01-(table_x[i]-x)
                y_above_ubound = y -␣
↪(table_y[i]+(table_x[i]-x)*table_y[i]*table_x[i])
            if (y_above_ubound<0 and y_above_lbound<0) or (y<np.exp(-x**2/2)):
                return x*sign, state
    for n in range(size):
        n1, state = variate(state)
        output.append(mu+sigma*n1)
    return output, state
```

3.5 Testing

Below we will conduct some tests to see whether our functions work in practice.

```
print('R')
print('Variates from rnorm:')
robjects.r('''
set.seed(1)
print(rnorm(5))
''')
print('Variates from our function:')
r_random_normal(r_state_from_seed(1), size=5)[0]
print('\nC++')
print('Variates from the boost random library:')
n_c1 = cplusplus_boost_normal(cplusplus_state_from_seed(1), size=5)[0]
print(n_c1)
```

```
R
Variates from rnorm:
```

```
[1] -0.6264538  0.1836433 -0.8356286  1.5952808  0.3295078
Variates from our function:

C++
Variates from the boost random library:
[2.2849294606911874, -0.6686271260558455, 0.006806818552631442,
0.26210960720511955, -0.8068316573936928]
```

4 Appendix

4.1 Ziggurat Method

This is a method to generate samples from decreasing densities. It uses a form of rejection sampling to generate its variates and works in the following way.

We choose a covering set ($\mathcal{Z}$) for the area ($\mathcal{C}$) under a density $f(x)$ which consists of a set of rectangles of equal area (v) stacked on top of each other, with the bottom strip tailing off to infinity. We will label the right most co-ordinate of rectangle i (R_i) by x_i. So we have that $0 = x_0 < x_1 < x_2 < \ldots$

If a random rectangle R_i is selected then a random point in R_i is Ux_i with U uniform $(0, 1)$, and if $x < x_{i-1}$ then our random point (x, y) must be in $\mathcal{C}$ and so we can confirm the point x without having to calculate y.

Let r be the rightmost x_i. We may generate from the base strip as follows:

- generate $x = \frac{vU}{f(r)}$, with U uniform $(0, 1)$
- If $x < r$, return x
- Else return x from the tail

So we get an x from the base rectangle with probability $\frac{rf(r)}{v}$, the same as generating an x from one of the other rectangles. This ensures that we can easily sample an x from $\mathcal{Z}$ as we can randomly choose a rectangle according to a uniform distribution (as they can be chosen with equal probability) and then we can easily sample from the corresponding rectangle.

Python implements a version of this algorithm that uses 255 rectangles, and a base strip as the covering set.

To apply the algorithm in its entirety we can use the following procedure, which uses the output of a 32-bit word generator for maximum efficiency.

1. Generate a random 32-bit word j, let i be the index provided by the rightmost 8 bits of j.
2. Set $x = jw_i$. If $j < k_i$ return x
3. If $i = 0$ return an x from the tail
4. If $[f(x_{i-1} - f(x_i))]U < f(x) - f(x_i)$, return x
5. Go to step 1

Here $w_i = 2^{31}x_i$ and $k_i = \lfloor 2^{32}(\frac{x_{i-1}}{x_i}) \rfloor$ for $1 \leq i \leq 255$ and for $i = 0$ $k_0 = \lfloor 2^{32}\frac{rf(r)}{v} \rfloor$ and $w_0 = 2^{31}\frac{v}{f(r)}$

So for each density, we consider we need to find the appropriate values of r and v, and consequently x_i, to form our rectangles for our covering set $\mathcal{Z}$. We can then sample from $\mathcal{Z}$ and reject samples according to our algorithm to generate variates of our desired distribution. The rejection rate for this algorithm is very low making it efficient (for most distributions a sample x is accepted around 98% of the time).