Supporting Information: A Simple Algorithm for Converting Random Number Generator Outputs to Universal Distributions to Aid Teaching and Research in Modern Physical Chemistry

Barry Y. Li,[1*] Tim Duong,[1*] Daniel Neuhauser,[1] Anastassia N. Alexandrova,[1,2] and Justin R. Caram[1]

[1]*Department of Chemistry and Biochemistry, University of California, Los Angeles, Los Angeles, CA, 90095, USA*
[2]*Department of Materials Science and Engineering, University of California, Los Angeles, Los Angeles, CA, 90095, USA*

Contents

1. The presented RNC in MATLAB and Python with comments

MATLAB:

```
clear, clc, clf
%% simple numerical random number converter from inv(CDF)
%% Barry Y. Li and Tim Duong (2024)
%% in the "Workspace", the output new_vec is the array of rand #'s you want

lower = 0;                                          % sampling range lower bound
upper = 16;                                         % sampling range upper bound
%upper = pi.^2./4;
n = 1e7;                                            % number of draw random #'s
randy = rand(n,1);
x = lower:(1e-4):upper;

% please input f as a known pdf ------------------------------------------
%f = exp(-x);                                       % single-exponential decay
%f = 1d0./sqrt(pi).*exp(-(x-3).^2d0);               % gaussian distribution
%f = 2d0./pi.*sin(2d0.*sqrt(x));                    % a sine-based distribution
F = x.^2.*exp(-x./2).*(sin(x)).^2;                  % polynomial+exponential+sine
```

```
%% ############ you do not have to change anything below :-) ##############

% this is before normalization ----------------------------------------------
for i = 1:(length(x)-1d0)
    intf(i) = 1d0./2d0.*(f(i+1)+f(i)).*(x(i+1)-x(i));% trapezoidal integral
end
cdf_f = 0d0;
for i = 1:(length(x)-1d0)
    cdf_f(i+1) = cdf_f(i)+intf(i);                % get the CDF numerically
end
% --------------------------------------------------------------------------

% this is after normalization -----------------------------------------------
f = f./cdf_f(end);
for i = 1:(length(x)-1d0)
    intf(i) = 1d0./2d0.*(f(i+1)+f(i)).*(x(i+1)-x(i));% trapezoidal integral
end
cdf_f = 0d0;
for i = 1:(length(x)-1d0)
    cdf_f(i+1) = cdf_f(i)+intf(i);                % get the CDF numerically
end
% --------------------------------------------------------------------------

figure(1)
yyaxis left
plot(x,f,'LineWidth',2)
ylabel('PDF(x)')
yyaxis right
plot(x,cdf_f,'LineWidth',2)
xlabel('x')
ylabel('CDF(x)')
xlim([lower upper])
ylim([0 1.1])
box on
set(gca,'linewidth',2);
set(gca,'fontsize',16);

for i = 1:n
    c(i) = closest_value(cdf_f, randy(i));
end
new_vec = x(c);

figure(2)
histogram(new_vec,100,'LineWidth',1.36,'EdgeColor','b','FaceAlpha',0)
xlim([lower upper])
xlabel('x')
ylabel('Counts')
box on
set(gca,'linewidth',2);
set(gca,'fontsize',16);


%% function 1: trapezoidal integral -------------------------------------
function nig = numint(x,f,i)
    nig = trapz([x(1):x(i)],[f(1):f(i)]);
```

```
end

%% function 2: binary search subroutine ---------------------------------
function v = closest_value(y,x)
findind = 1d0;
endind = length(y);

% binary search for index
while ((endind - findind) > 1d0)
    midind = floor((endind+findind)./2d0);
    if (y(midind) >= x)
        endind = midind;
    else
        findind = midind;
    end
end
if ((endind-findind) == 1d0) && (abs(y(endind)-x) < abs(y(findind)-x))
    findind = endind;
end
v = findind;

end
```

Python:

- To get the random numbers stored in a *.log* file, example:
  python3 build_rnc.py new_vec.log &

- To obtain the histogram of the random numbers, example:
  python3 build_hist.py new_vec.log &

```
## build_rnc.py
## simple numerical random number converter from inv(CDF)
## Barry Y. Li and Tim Duong (2024)

import sys
import numpy as np

def generate_random_numbers(filename):
    # 1. parameters
    lower = 0                               # sampling range lower bound
    upper = 16                              # sampling range upper bound
    n = int(1e7)                            # number of random numbers to draw
    randy = np.random.rand(n)
    x = np.arange(lower, upper + 1e-4, 1e-4)

    # 2. define PDF
    def f(x):
        return x**2 * np.exp(-x / 2) * (np.sin(x))**2

    # 3. trapezoidal integral function
    def numint(x, f, i):
        return np.trapz(f[:i+1], x[:i+1])
```

```
    # 4. compute cdf numerically
    intf = np.zeros(len(x) - 1)
    for i in range(len(x) - 1):
        intf[i] = 0.5 * (f(x[i+1]) + f(x[i])) * (x[i+1] - x[i])

    cdf_f = np.cumsum(intf)
    # 5. normalize the cdf
    f = f(x) / cdf_f[-1]
    intf = np.zeros(len(x) - 1)
    for i in range(len(x) - 1):
        intf[i] = 0.5 * (f[i+1] + f[i]) * (x[i+1] - x[i])
    cdf_f = np.cumsum(intf)

    # 6. convert random numbers based on the cdf
    def closest_value(y, x):
        findind = 0
        endind = len(y) - 1
        while endind - findind > 1:
            midind = (endind + findind) // 2
            if y[midind] >= x:
                endind = midind
            else:
                findind = midind
        if endind - findind == 1 and abs(y[endind]-x) < abs(y[findind]-x):
            findind = endind
        return findind

    new_vec = np.zeros(n)
    for i in range(n):
        c = closest_value(cdf_f, randy[i])
        new_vec[i] = x[c]

    # 7. Write new_vec to a .log file
    np.savetxt(filename, new_vec)

    print(f"Random numbers saved to {filename}.")

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: python3 rng.py <filename>")
    else:
        filename = sys.argv[1]
        generate_random_numbers(filename)
```

```
## build_hist.py
## histogram plotting function
## Barry Y. Li and Tim Duong (2024)

import sys
import numpy as np

def plot_histogram(filename):
    # read data from the file
```

```
    try:
        data = np.loadtxt(filename)
    except FileNotFoundError:
        print(f"Error: File '{filename}' not found.")
        return

    # calculate histogram
    hist, bin_edges = np.histogram(data, bins=100)

    # find maximum count for scaling
    max_count = np.max(hist)

    # display histogram in terminal (vertical)
    print("Histogram of data:")
    for i in range(len(hist)):
        bin_str = f"{bin_edges[i]:.2f} - {bin_edges[i+1]:.2f}"
        count_str = "#" * int(hist[i] * 50 / max_count)
                    # scale the counts to fit in 50 characters
        print(f"{bin_str.ljust(15)}| {count_str}")

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: python3 histo.py <filename>")
    else:
        filename = sys.argv[1]
        plot_histogram(filename)
```

2. Single-exponential decay distribution with analytical CDF and inv(CDF)

As derived in the main text, the analytical form of the CDF for the probability density function, PDF$(x) = ke^{-kx}$, where k is a real number, is $1 - e^{-kx}$. Therefore, the random number can be drawn directly from its inverse (i.e. $x = -\frac{1}{k}\ln\xi$, where ξ is a uniformly-distributed random vector between 0 and 1). The MATLAB script is shown below for the $k = 1$ condition with its results plotted in Figure S1.

```
clear,clc,clf

x = 0d0:0.001:10d0;
pdf = exp(-x);
cdf = 1-exp(-x);
n = 1e5;
v = -log(rand(n,1));

figure(1)
yyaxis left
plot(x,pdf,'b','LineWidth',2.6)
hold on
plot(x,cdf,'b','LineWidth',2.6)
hold off
yyaxis right
histogram(v)
```

```
legend('PDF','CDF','Rand #')
box on
set(gca,'linewidth',2);
set(gca,'fontsize',13.6);
```

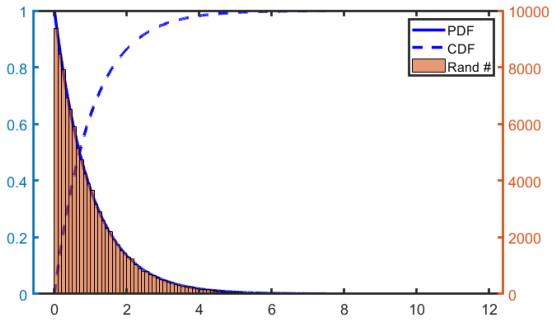

Figure S1 PDF, analytical CDF, and random numbers in the form of a histogram of 10^5 numbers obtained from the analytical inverse CDF of a single-exponential decay distribution in the $k = 1$ case.

3. Marsaglia polar method which utilizes the Box-Muller transform

The Marsaglia polar method is a common way of converting random numbers into a Gaussian distribution. It utilizes the coordinate transformation from cartesian to polar, where

$$z_1 = \sqrt{-2\ln u_1} \cdot \cos(2\pi u_2) \qquad \text{(eq. S1)}$$

and

$$z_2 = \sqrt{-2\ln u_1} \cdot \sin(2\pi u_2). \qquad \text{(eq. S2)}$$

Here, u_1 and u_2 are uniformly-distributed random numbers between 0 and 1, and z_1 and z_2 are sets of Gaussian-distributed random numbers. The resulting random numbers, z, is a union set of z_1 and z_2 with a Γ_g (i.e. Gaussian FWHM) parameter, where

$$z = \frac{\Gamma_g}{2\sqrt{2\ln 2}} \cdot \{z_1 \cup z_2\}. \qquad \text{(eq. S3)}$$

This transformation is a manifestation of the Central Limit Theorem (CLT), which states that the sum (i.e. average) of a large number of independent, identically distributed random variables, regardless of their original distribution, tends to follow a normal distribution.(Marsaglia et al., 1964) In this case, the transformation ensures that z_1 and z_2 are independent and identically distributed. Their distribution approximates the standard normal distribution as shown in Figure S2. The MATLAB script is provided below.

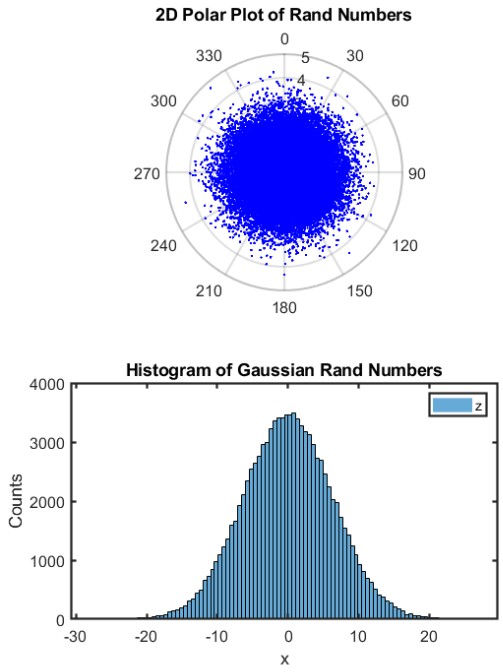

Figure S2 Upper: Polar plot of the generated 10^5 random numbers with $\Gamma_g = 15$ as they are bounded in a 2D circle with a 2D Gaussian shape. Lower: Histogram of the 10^5 random numbers that reproduces the desired Gaussian PDF.

```
clear,clc,clf
n = 1e5;
sigma = 15; % Gaussian FWHM

u1 = rand(floor(n./2),1);
u2 = rand(floor(n./2),1);
z1 = sqrt(-2d0.*log(u1)).*cos(2.*pi.*u2);
z2 = sqrt(-2d0.*log(u1)).*sin(2.*pi.*u2);
z = [z1; z2];
z = sigma./(2d0.*sqrt(2d0.*log(2))).*z;

figure(1)
subplot(2,1,1)
polarplot(atan2(z2,z1),sqrt(z1.^2 + z2.^2),'b.');
title('2D Polar Plot of Rand Numbers');
ax = gca;
ax.ThetaZeroLocation = 'top';
ax.ThetaDir = 'clockwise';
set(gca,'linewidth',1.6);
set(gca,'fontsize',12);

subplot(2,1,2);
histogram(z,100);
title('Histogram of Gaussian Rand Numbers');
xlabel('x');
ylabel('Counts');
legend('z');
```

```
box on
set(gca,'linewidth',1.6);
set(gca,'fontsize',12);
```

4. Analytical solutions to the cyclic chemical reaction (F1-ATPase model)

Given that the mean time for a single-step $\tau = 1/k[\mathrm{ATP}]$, and the distribution of time to be

$$p_1(t) = e^{-t/\tau}/\tau, \tag{eq. S4}$$

the two-step probability distribution of time is a convolution, where

$$\begin{aligned} p_2(t) = p_1(t') * p_1(t-t') &= \frac{1}{\tau^2}\int_0^t e^{-t'/\tau} e^{-(t-t')/\tau}\, dt' \\ &= \frac{e^{-t/\tau}}{\tau^2}\int_0^t dt' = \frac{t}{\tau^2}e^{-t/\tau}. \end{aligned} \tag{eq. S5}$$

Similarly, the three-step probability distribution of time is

$$\begin{aligned} p_3(t) = p_2(t'') * p_1(t-t'') &= \frac{1}{\tau^3}\int_0^t t'' e^{-t''/\tau} e^{-(t-t'')/\tau}\, dt'' \\ &= \frac{e^{-t/\tau}}{\tau^3}\int_0^t t''\, dt'' = \frac{t^2 e^{-t/\tau}}{2\tau^3}. \end{aligned} \tag{eq. S6}$$

This result can be generalized to an n-step process, where

$$p_n(t) = \frac{e^{-t/\tau}}{\tau^n}\cdot\frac{t^{n-1}}{(n-1)!} \tag{eq. S7}$$

(i.e. a typical Poisson distribution).

To analytically calculate the mean time for the full cycle (i.e. from i state back to state i), we simply need to evaluate the expectation value of $p_3(t)$. Therefore,

$$\langle t\rangle_{\mathrm{cycle}} = \int_0^\infty t\cdot p_3(t)\, dt = \frac{1}{2\tau^3}\int_0^\infty t^3 e^{-t/\tau}\, dt = \frac{1}{2\tau^3}\cdot 6\tau^4 = 3\tau. \tag{eq. S8}$$

5. MATLAB script for the retrospective MZI Fourier spectroscopy simulation

(Atallah et al., 2019)

```
clear,clc,clf
%% Simple retrospective MZI Fourier spectroscopy simulation
%% Barry Y. Li and Tim Duong (2024)
%% Parameter Box ##################################################################
delta = -5e-5:2.5e-8:5e-5;                              % stage position (meter)
emitter_r = [1.25 0.40 0.16 -6];                 % red {E0,Gamma_g,Gamma_l,beta}
emitter_b = [1.35 0.18 0.08 -3];                % blue {E0,Gamma_g,Gamma_l,beta}
fi = 1.0d0;                                         % energy (E) RNC start point
ff = 1.5d0;                                           % energy (E) RNC end point
%% ################################################################################
```

```
%% ############ you do not have to change anything below :-) ##############
tic
avg_time = 1e5;                                     % average life time (ps)
pulse_space = avg_time.*exp(1);                % pulse in-between time (ps)
photon_per_pulse = 0.05;                       % average # photons per pulse
n_photon_goal = 1e4;                            % desired # photon to detect
n_pulse = 1./photon_per_pulse.*n_photon_goal;

for i = 1:n_pulse
    pulse_t(i) = (i-1).*pulse_space;
end
measure_time_in_s = pulse_t(end)./1e12;

for i = 1:length(delta)
    intere(i,:,:) = spec(delta(i),avg_time,pulse_space,photon_per_pulse,...
        n_photon_goal,emitter_r,emitter_b,fi,ff,n_pulse,pulse_t);
end

figure(1)
for i = 1:length(delta)
    cum_count(i) = sum(intere(i,2,:));
end

[earray,P2] = fft_spec(delta,cum_count);


figure(1)
plot(delta'.*1e6,cum_count'./max(max(cum_count')),'Color',...
    [0 0.4470 0.7410],'LineWidth',1.6)
xlim([-50 50])
ylim([-1.1 1.1])
xlabel('Stage Position (um)')
ylabel('Signal S_A-S_B')
title('Dual-emitter Interferogram')
box on
set(gca,'fontsize',16);
set(gca,'linewidth',1.6);

figure(2)
plot(earray,P2/max(P2),'k:','LineWidth',2.6)
xlim([fi ff])
ylim([0 1.1])
xlabel('Energy (eV)')
ylabel('Norm. Intensity')
title('Dual-emitter FFT Spectrum')
box on
set(gca,'fontsize',16);
set(gca,'linewidth',1.6);
toc


%% Core simulation functions ###############################################
%% -------------------------------------------------------------------------
```

```
%% 1. Photon (pulsed) stream simulation - Sub-poisson (antibunching)
function ps = spec(delta,avg_time,pulse_space,photon_per_pulse,...
        n_photon_goal,emitter_r,emitter_b,fi,ff,n_pulse,pulse_t)
c = 299792458;
hbar = 1.054571817e-34;

detc = 1d0./2d0.*erfc(1000d0.*(rand(n_pulse,1)-photon_per_pulse));
pulse_with_photon = pulse_t(find(detc == 1));
tau(:) = -avg_time.*log(rand(length(pulse_with_photon),1));

T = pulse_with_photon + tau;
T(end) = [];

np = length(T);

%% ------------------------------------------------------------------------
%% 2. Now call out all the frequency-domain info
fd = (ff - fi)./(np - 1d0);
x = 1.0:fd:1.5;
fr = simu_spec(emitter_r,x);
fb = simu_spec(emitter_b,x);
cdf_r = cdf_gen(x,fr);
cdf_b = cdf_gen(x,fb);
randyr = rand(np,1);
randyb = rand(np,1);
for i = 1:np
    [vr(i) cr(i)] = closest_value(cdf_r, randyr(i));
    [vb(i) cb(i)] = closest_value(cdf_b, randyb(i));
end
new_vecr = x(cr);
new_vecb = x(cb);
new_vec = [new_vecr new_vecb]';

%% ------------------------------------------------------------------------
%% 3. The result will be (t,omega)
omega = new_vec.*1.602176565e-19./hbar;
prob_a = 1d0./2d0.*(1+cos(delta.*omega./c));
prob_b = 1d0./2d0.*(1-cos(delta.*omega./c));

for i = 1:length(prob_a)
    detc_choice(i) = detector(prob_a(i));
end
count_array = round(detc_choice(:));

deteca = sum(count_array);
detecb = length(prob_a) - deteca;


[counts_a,centers_a] = histcounts(tau,200);
counts_a(numel(centers_a)) = 0d0;
ps = [centers_a; (deteca-detecb).*counts_a];
end


%% dependency functions ###################################################
```

```
%% 1. ---------------------------------------------------------------------
function build_cdf = cdf_gen(x,f)
% this is before normalization:
for i = 1:(length(x)-1d0)
    intf(i) = 1d0./2d0.*(f(i+1)+f(i)).*(x(i+1)-x(i));         % trapezoidal
end
cdf_f = 0d0;
for i = 1:(length(x)-1d0)
    cdf_f(i+1) = cdf_f(i)+intf(i);                % get the CDF numerically
end

% this is after normalization:
f = f./cdf_f(end);
for i = 1:(length(x)-1d0)
    intf(i) = 1d0./2d0.*(f(i+1)+f(i)).*(x(i+1)-x(i));% trapezoidal integral
end
cdf_f = 0d0;
for i = 1:(length(x)-1d0)
    cdf_f(i+1) = cdf_f(i)+intf(i);                % get the CDF numerically
end
build_cdf = cdf_f;
end

%% 2. ---------------------------------------------------------------------
function spec = simu_spec(data,xa)
    mu  = [data(1)];
    gba = [data(2)];
    lba = [data(3)];
    gsk = [data(4)];

    f1 = g(xa,mu(1),gba(1)./(2.*sqrt(2.*log(2))),gsk(1),mu(1),lba(1)./2d0);

    f1n = f1./max(f1);
    spec = f1n;
end

%% 3. ---------------------------------------------------------------------
function spdf = g(x,mean_g,std_g,skewness,mean_l,w_l)
    % gaussian normalization constant
    gauss_norm = 1d0./(std_g.*sqrt(2d0.*pi));

    % skewed Gaussian
    gauss_pdf = gauss_norm.*exp(-1d0./2d0.*((x-mean_g)./std_g).^2d0);
    gauss_pdf = gauss_pdf.*(1+erf(skewness.*(x-mean_g)./(std_g.*sqrt(2))));

    % lorentzian PDF
    lorentz_pdf = 2d0./pi.*(w_l./(2d0.*((x-mean_l).^2d0+(w_l./2).^2)));

    % compute the product of the two distributions
    spdf = gauss_pdf.*lorentz_pdf;
end

%% 4. ---------------------------------------------------------------------
function [v,inf] = closest_value(arr,val)
len = length(arr);
```

```
inf = 1;
sup = len;
while sup-inf > 1
    med = floor((sup+inf)./2d0);
    if arr(med) >= val
        sup = med;
    else
        inf = med;
    end
end
if sup-inf == 1 && abs(arr(sup)-val) < abs(arr(inf)-val)
    inf = sup;
end
v = arr(inf);
end

%% 5. ---------------------------------------------------------------------
function detc = detector(a)
    x = rand(1);
    detc = 1d0./2d0.*erfc(1000d0.*(x-a));
end

%% 6. ---------------------------------------------------------------------
function [earray,P2] = fft_spec(delta,cum_count)
L = length(cum_count);
X = cum_count;
Y = fft(X);
P2 = abs(Y/L);

ddelta = mean(diff(delta))*1e2;                                    % in cm
ndelta = length(delta);

k_max = 1d0./(ddelta);                                              % in cm-1
emax = k_max.*1.2398e-4;                                           % now in eV
k_min = 1d0./(ndelta.*ddelta);                                      % in cm-1
emin = k_min.*1.2398e-4;                                           % now in eV

earray = emin:((emax-emin)./(ndelta-1)):emax;
earray = earray - emin;                                        % spectral shift
end
```

6. References

- Atallah TL, Sica A V., Shin AJ, Friedman HC, Kahrobai YK, Caram JR. Decay-Associated Fourier Spectroscopy: Visible to Shortwave Infrared Time-Resolved Photoluminescence Spectra. Journal of Physical Chemistry A 2019;123:6792–8. https://doi.org/10.1021/acs.jpca.9b04924.
- Marsaglia G, Bray TA, Summary O. A CONVENIENT METHOD FOR GENERATING NORMAL VARIABLES. vol. 6. 1964.