Homework 2

Andrew Loeppky

Phys 509

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

from scipy.special import factorial as fac
from scipy.stats import norm
from scipy.stats import binom
from scipy.stats import poisson
from scipy.optimize import minimize
```

Question 1

Here are three easy applications of Bayes' theorem:

A. Suppose the rate R of visible galactic supernovae is unknown but that supernovae follow a Poisson distribution. In the past 10 centuries astronomers have observed four supernovae in our galaxy. Assuming a uniform prior for the rate R, use Bayes' theorem to calculate the probability distribution for R. Now repeat the calculation, assuming this time that the prior for R is uniform in log10 R (i.e. it's equally probable that the true rate is between 0.02 and 0.2 as it is that it is between 0.2 and 2.0). Plot the resulting posterior probability distribution for R in both cases.

$$p(R|D,I) = \frac{p(R|I)p(D|R,I)}{p(D|I)}$$

Choose bounds for R between 20 in 10 centuries and 1/100 centuries. Adopt a timescale of 1000 years for all calculations. Prior for R is either uniform:

$$p(R|I) = \frac{1}{\Delta R}; \quad 0.1 < R < 20,$$

$$\frac{1}{\Delta R} = \frac{1}{R_{max} - R_{min}}$$

or uniform in $log_{10}R$ (Jeffrey's Prior):

$$p(R|I) = \frac{1}{Rln(R_{max}/R_{min})}$$

Now calculate the likelihood, with our observation of $k = 4$ supernova in 1000 years

$$p(D = k|R, I) = \frac{1}{k!} R^k e^{-R}$$

Finally, sum over all possible R to get the global likelihood

$$p(D|I) = \sum_{R_{min}}^{R_{max}} \frac{1}{R_{max} - R_{min}} \frac{1}{k!} R^k e^{-R}$$

or

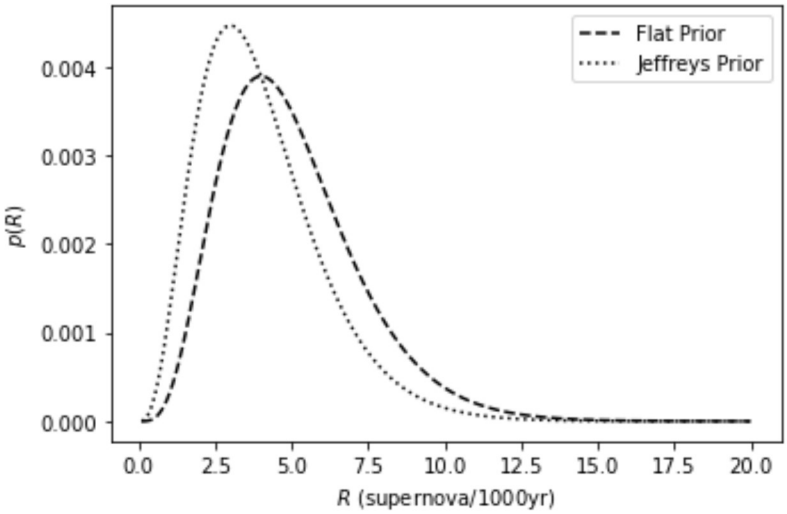
$$p(D|I) = \sum^{R_{max}} \frac{1}{\quad} \frac{1}{k!} R^k e^{-R}$$

In [2]:
```
### prior probability p(R|I) ###

Rmin = 1 / 10  # supernova per 1000 years
Rmax = 20

# create 2 different R's, just for plot clarity
R = np.linspace(Rmin, Rmax, 1000)

# calculate each prior
un_prior = 1 / (Rmax - Rmin)
jf_prior = 1 / (R * np.log(Rmax/Rmin))
```

In [3]:
```
### sampling probability p(D|R,I) ###

k = 4  # observed supernova per 1000 years
lkhd = 1 / fac(k) * R ** k * np.exp(-R)  # poisson dist
```

In [4]:
```
### global likelihood p(D|I) ###

glbl_flat = np.sum(un_prior * lkhd)
glbl_jeff = np.sum(jf_prior * lkhd)
```

In [5]:
```
### calculate the two distns for R and plot ###

flat = un_prior * lkhd / glbl_flat
jeff = jf_prior * lkhd / glbl_jeff

plt.plot(R, flat, color="k", linestyle="--", label="Flat Prior")
plt.plot(R, jeff, color="k", linestyle=":", label="Jeffreys Prior")
plt.xlabel("$R$ (supernova/1000yr)")
plt.ylabel("$p(R)$")
plt.legend();
```



B. Measurements are drawn from a uniform distribution spanning the interval (0, m). The probability of getting a measurement outside of this range is zero. The endpoint m is not well-known, but a prior experiment yields a Gaussian prior of m = 3 +/- 1. You take three measurements, getting values of 2.5, 3.1, and 2.9. Use Bayes' theorem to calculate and plot the new probability distribution for m.

$$p(m|D,I) = \frac{p(m|I)p(m|D,I)}{p(D|I)}$$

Prior for m:

$$p(m|I) = \frac{1}{\sqrt{\sigma^2 2\pi}} e^{\frac{-(m-\bar{m})^2}{2\sigma^2}}$$

Properties of a uniform distn on a, m:

$$f(x) = \begin{cases} \frac{1}{m-a} & a \leq x \leq m \\ 0 & else \end{cases} \qquad (G5.40^*)$$

$$mean = \frac{a+m}{2}$$

Given $a - 0$, we want to solve for m, subbing the mean of our dataset x_1, x_2, x_3 on the LHS

$$m_{mcas} = 2\bar{x}$$

$$\sigma_{mmeas} = \sigma_{\bar{x}} \frac{\partial m_{mcas}}{\partial \bar{x}} = 2\sigma_{\bar{x}}$$

The likelihood function for m must be a gaussian, because it depends on the sum of independent random variables. (Or nearly gaussian, we saw in assignment 1 precisely what the sum of 3 random variables drawn from a uniform distribution looks like)

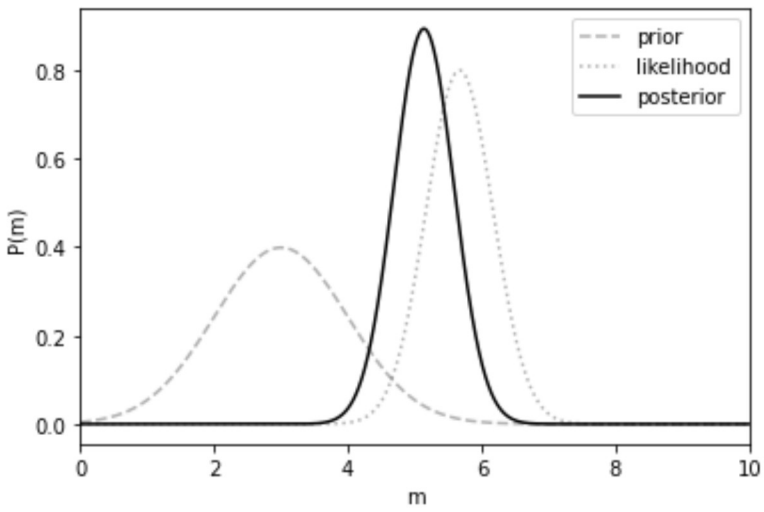
$$p(m|D,I) = norm(2\bar{x}, 2\sigma_x)$$

Integrate over m to get global likelihood

$$p(D|I) = \int_0^\infty norm(3,1) \cdot norm(2\bar{x}, 2\sigma_x) \cdot dm$$

In [6]:
```
deltax = 1e-4 # numerical resolution
mmax = 50 # max upper coord
data = np.array([2.5, 3.1, 2.9]) # measured data

# prior
m_guess = 3
m = np.arange(0, mmax, deltax)
m_prior = norm.pdf(m, loc=m_guess, scale=1)

# likelihood
xbar = np.mean(data)
sigma_x = np.std(data)
m_lkhd = norm.pdf(m, loc=(2 * xbar), scale=(2 * sigma_x))

# global
m_glbl = np.sum(m_prior * m_lkhd) * deltax

# plug into bayes theorem
m_post = m_prior * m_lkhd / m_glbl
```

In [7]:
```
# plot result
plt.plot(m,m_prior, "k", linestyle="--", alpha=0.3, label="prior")
plt.plot(m,m_lkhd, "k", linestyle=":", alpha=0.3, label="likelihood")
plt.plot(m,m_post, "k", label="posterior")
plt.xlim(0,10)
plt.legend()
plt.xlabel("m")
plt.ylabel("P(m)");
```



C. Suppose that an unmanned rocket is being launched, and that at the time of the launch a certain electronic component is either functioning or not functioning. In the control centre there is a warning light that is not completely reliable. If the electronic component is not functioning, the warning light goes on with probability 1/2; if the component is functioning, the warning light

goes on with probability 1/3. At the time of launch, the operator looks at the light and must decide whether to abort the launch. If she aborts the launch when the component is functioning well, she wastes $2M. If she doesn't abort the launch but the component has failed, she wastes $ 5M. If she aborts the launch when the component is malfunctioning, or if she lets the launch proceed when the component is working normally, there is no cost. Suppose that the prior probability of the component failing is 2/5. During launch the warning light doesn't go on. From a costs standpoint, should she abort the mission or not? Compute and compare the expected cost of launching to the expected cost of aborting, given that the light didn't go on.

Case 1: Abort Launch

First, compute the posterior probability that the component is working c, given the fact that there is no light. The prior probability of the component failing is 2/5, so the probability of it not failing is:

$$p(c|I) = \frac{3}{5}$$

The probability of data (the light not coming on), given that the component is working:

$$p(D|c, I) = \frac{2}{3}$$

And the global probability of "no light" is:

$$p(D|I) = p(D|c) + p(D|\bar{c}) = \left(\frac{3}{5}\frac{2}{3}\right) + \left(\frac{2}{5}\frac{1}{2}\right) = \frac{3}{5}$$

The posterior probability times the expected cost of aborting the launch:

$$p(c|D, I) = \frac{3/5 \cdot 2/3}{3/5} \cdot \$2M = \$1.33M$$

Case 2: Go for it

Now, compute the posterior for $\bar{c}$, given the same information. Prior:

$$p(\bar{c}|I) = \frac{2}{5}$$

Likelihood:

$$p(D|\bar{c}, I) = \frac{1}{2}$$

global probability is the same $3/5$ in both cases. Calculate the expected cost of launching:

$$p(\bar{c}|D, I) = \frac{2/5 \cdot 1/2}{3/5} \cdot \$5M = \$1.67M$$

So the best decision here would be to abort the launch and use the savings to re-design rocket

parts with less than a 2/5 chance of failure.

Question 2

A COVID-19 study submitted for publication in April 2020 tested 3330 people in Santa Clara County, California for COVID-19 antibodies, and found 50 positive test results. To determine the false positive rate of the test, the researchers tested samples of blood from a control group of people who were known to have never been exposed. In this control group, 16 out of 3324 samples yielded a false positive result. To determine the sensitivity (false negative rate) of the test, another control group of confirmed cases was tested, with 130 out of 157 yielding a positive result (i.e. 27 false negative results). Calculate the Bayesian 95% central interval on the fraction of people in Santa Clara County who actually had antibodies for COVID-19, marginalizing over the false positive and false negative rates. Assume flat priors on all parameters. Submit a plot of the posterior distribution for the true incidence rate as well as your code or calculation.

Consider a larger sample of $k = 10000$, representative of the population of Santa Clara county. Call the real number of people with covid antibodies R, the number of false positive instances f_+, and the number of false negatives f_-. Write Bayes theorem for 3 variables R, f_+, f_-

$$P(R|D,I) = \frac{\iint df_- df_+ P(f_+|I)P(f_-|I)P(R|I)P(D|f_-,f_+,R,I)}{P(D|I)}$$

Priors

Instances of actual covid antibodies:

$$P(R|I) = \frac{1}{k} \; on \; (1,k)$$

False positives (Ideally would be on $(1,R)$, but we dont know R for certain, other than it is $\leq k$):

$$P(t_+|I) = \frac{1}{k} \; on \; (1,k)$$

Similarly for false negatives (ideally on $(1, k-R)$):

$$P(t_-|I) = \frac{1}{k} \; on \; (1,k)$$

Likelihood

We expect each variable to be poisson distributed, as insances of covid antibodies, false positives, and false negatives are "rare", and there is no possibility of double counting -- each test can only have one result, and that result can either be accurate or not.

Poisson dist:

$$P(n|\lambda) = \frac{\lambda^n e^{-\lambda}}{n!}$$

> R is poisson distributed with $\lambda_R = 50/3330 \approx 0.015$ and $n = k$
>
> f_+ is poisson, but depends on R -- the number of false positives depends on the number of positive tests, and cannot be greater than it. $\lambda_{f+} = 16/3324 \approx 0.0048$ and $n = R$
>
> f_- is similar, but depends on the number of negative tests. $\lambda_{f-} = 27/157 \approx 0.17$ and $n = (k - R)$

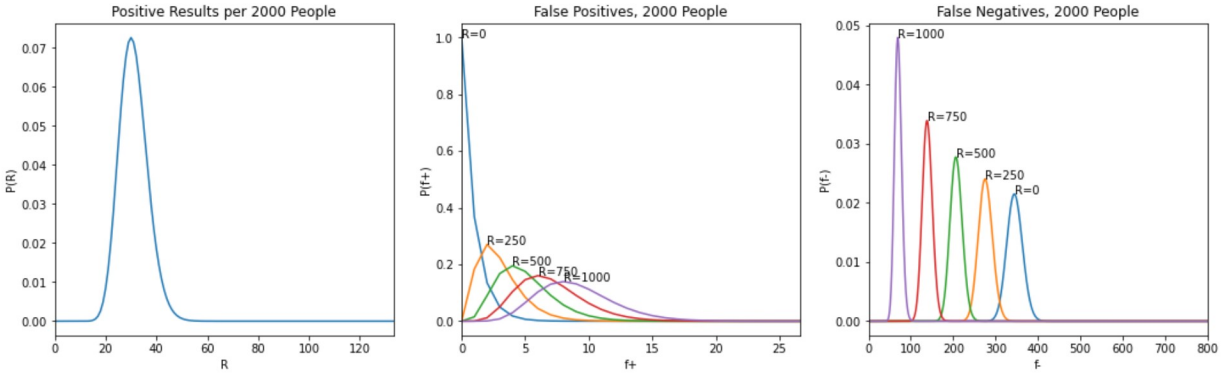
Define a model -- the true number of cases is the number measured, minus false positives, plus false negatives

$$P(D|R, f_-, f_+) = R - f_- + f_+$$

Now, create the probability field over a population of k people, and generate probabilities for f_- and f_+ for each value of R

In [8]:

```
k = 2000 # population
numpeople = np.arange(k)

### priors ###
prior_R = 1 / k
prior_fpos = 1 / k
prior_fneg = 1 / k

### likelihood ###
lmda_R = int(round(50 / 3330 * k))  # expected number of pos tests
lmda_fpos = 16 / 3324  # expected number of false positives
lmda_fneg = 27 / 157  # expected number of false negatives

# assign pdf for R
R = poisson.pmf(lmda_R, numpeople)

# assign the pdf of false positives, given R=i
pop_fpos = np.array([poisson.pmf(int(round(lmda_fpos * i)), numpeople) for i :

# assign the pdf of false negatives
pop_fneg = np.array([poisson.pmf(int(round(lmda_fneg * (k-i))), numpeople) fo

# do the plot
step = 400 # plotting parameter
fig, ax = plt.subplots(1,3, figsize=(18,5))

# plot R
ax[0].plot(R)
ax[0].set_xlim(0,step / 3)
ax[0].set_xlabel("R")
ax[0].set_ylabel("P(R)")
ax[0].set_title(f"Positive Results per {k} People")

# plot f+
[ax[1].plot(x) for x in pop_fpos[::step]]
[ax[1].annotate(f"R={i * 250}", (np.argmax(x), np.max(x))) for i, x in enumera
ax[1].set_xlim(0,step / 15)
ax[1].set_xlabel("f+")
ax[1].set_ylabel("P(f+)")
ax[1].set_title(f"False Positives, {k} People")

# plot f-
[ax[2].plot(y) for y in pop_fneg[::step]]
[ax[2].annotate(f"R={i * 250}", (np.argmax(y), np.max(y))) for i, y in enumera
ax[2].set_xlim(0,step * 2)
ax[2].set_xlabel("f-")
ax[2].set_ylabel("P(f-)")
ax[2].set_title(f"False Negatives, {k} People");
```



In [9]:

```
# fast numbers
pop = 10000
pos_tests = pop * 50 / 3330
false_pos = pos_tests * 16 / 3324
false_neg = (pop - false_pos) * 27 / 157
total_cases = pos_tests - false_pos + false_neg

print(f"positive tests:  {pos_tests}")
print(f"false positives: {false_pos}")
print(f"false negatives: {false_neg}")
print(f"total cases:     {total_cases}")
```

```
positive tests:  150.15015015015015
false positives: 0.7227444050548744
false negatives: 1719.6209293061372
total cases:     1869.0483350512325
```

Question 3

I recently set up a CO2 meter in my office. With the door and window closed, I measured the following readings vs time:

Time (hr:min)	CO2 level (ppm)
13:20	484
13:26	501
13:30	520
13:34	535
13:39	554
13:44	565
13:49	579
13:53	593
14:06	635
14:14	651
14:17	654

Assume that the data follows an exponential plateau, approaching some steady state value C. Do a maximum likelihood fit for this level, and determine the uncertainty on it. Note that you have not been given the uncertainties on the measured values---instead, assume that all measurements have the same uncertainty level, and fit for it as one of the parameters in your fit. Submit your code, the functional form you fit, and your result for the steady state value (with uncertainty). What uncertainty value per data point did you get? How much of that uncertainty can be attributed to the time binning (measurements are only reported to the nearest minute)?

In [10]:
```
co2 = pd.read_csv("co2.csv")
time = np.array([0,6,10,14,19,24,29,33,46,54,57]) # total elapsed time in mini
co2 = np.array(co2["CO2 (ppm)"])
```

ML equation:

$$-lnL(x_1,\ldots,x_n|\vec{\alpha}) = -\sum_{i=1}^{n} lnP(x_i|\vec{\alpha})$$

No special information about the errors or the device used to measure CO2, so assume gaussian errors. We then want to maximize this:

$$ln(L) = -\frac{1}{2}\sum_{i=1}^{N}\left(\frac{y_i - f(t_i|\alpha)}{\sigma_i}\right)^2 - \sum_{i=1}^{N} ln(\sigma_i\sqrt{2\pi})$$

Equation characterizing our system. c_0 is the ambient CO2 concentration, τ is a time constant, C is the steady state concentration above ambient:

$$f(t|c_o, C, \tau) = c_0 + C(1 - e^{-t/\tau})$$

"*Assume that all measurements have the same uncertainty level, and fit for it as one of the parameters in your fit*". So the final form of $ln(L)$ is:

$$ln(L) = -\frac{1}{2}\sum_{i=1}^{N}\left(\frac{y_i - (c_0 + C(1 - e^{-t_i/\tau}))}{\sigma}\right)^2 - Nln(\sigma\sqrt{2\pi})$$

In [44]:
```
def get_lnL(x, *args):
    '''
    defines -ln(L), to be fed into minimization routine
    x = np.array([C, tau, sig])
    *args = [co2, time]
    '''
    co2 = np.array(args[0])
    time = np.array(args[1])
    N = len(co2)
    #c0 = x[0]
    C = x[0]
    tau = x[1]
    sig = x[2]

    lnL = 0.5 * np.sum((co2 - co2[0] - C * (1 - np.exp(-time / tau)) / sig) *
          + (N * np.log(abs(sig) * np.sqrt(2 * np.pi)))

    return(lnL)
```

In [50]:
```
# do the minimization
# https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize

# guess at the values of c0, C, tau, sigma
x_guess = np.array([670, 60, 10])

minvals = minimize(get_lnL, x_guess, (co2, time), method='L-BFGS-B')
minvals
```

Out[50]:
```
      fun: 58.6685686078729
 hess_inv: <3x3 LbfgsInvHessProduct with dtype=float64>
      jac: array([0.01639221, 0.00071907, 0.03859242])
  message: 'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'
     nfev: 160
      nit: 31
     njev: 40
   status: 0
  success: True
        x: array([668.42170043, 133.59009676,   1.33596728])
```

Final fit parameters, which I have little faith in:

$$C = 668\ ppm$$

$$\tau = 134\ min$$

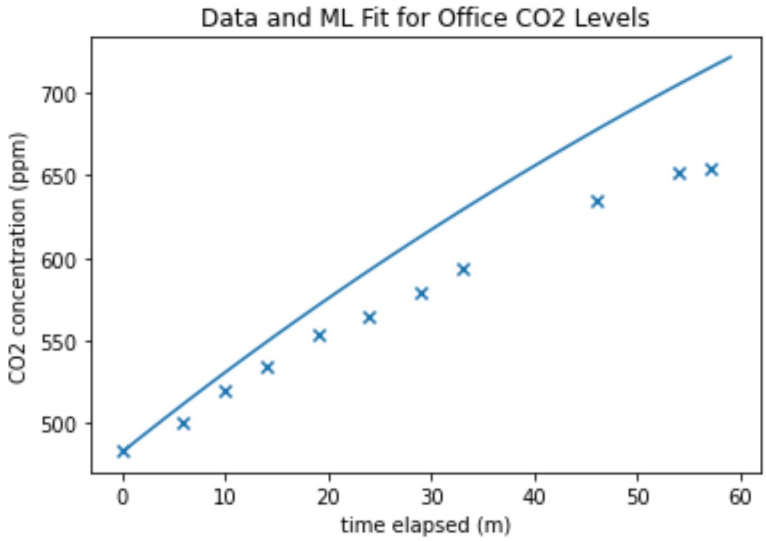
$$\sigma = 1.34\ ppm$$

Time Binning

Time binning essentially draws time values from a uniform distribution between the edges of the bin (variance $\Delta t^2/12$). The error in CO2 will depend on the width of the bin and $dCO2/dt$, the steepness of the function (they are uncorrelated). The error at any one CO2 measurement from time binning and measurement error:

$$\sqrt{\sigma_{CO2}^2 + \left(\frac{dCO2}{\quad}\right)^2 \sigma_{\quad}^2}$$

In [52]:
```
# plot the fit and the data
cmax = 668.42170043
tau = 133.59009676
x = np.arange(60)
fx = c0 + cmax * (1 - np.exp(-x / tau))

plt.plot(x, fx)
plt.scatter(time, co2, marker="x")
plt.xlabel("time elapsed (m)")
plt.title("Data and ML Fit for Office CO2 Levels")
plt.ylabel("CO2 concentration (ppm)");
```



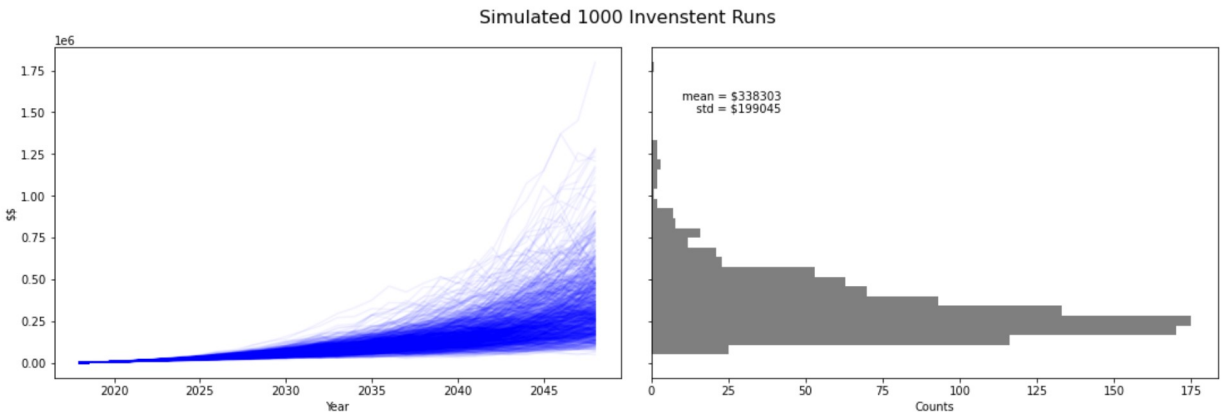
Question 4

In this problem you will simulate a simple retirement account consisting of up to three classes of investments.

A. The percentage yield on an investment has a Gaussian distribution with mean of 8% and SD 15%. (A yield of 8% would mean the amount of money increases by a factor of 1.08 in a year. A yield of -8% would mean multiplying by 0.92 instead.) Suppose that you put $3000 into a retirement account investing in this item on January 1st of every year, starting in 2018. What is the mean amount of money you will have in the account on Dec 31, 2047? Show a plot of the distribution of the amount of money on that date for 1000 trials of the "experiment". What is the SD? Hand in your code or equivalent documentation.

In [14]:
```
principal = 3000  # dollars
start_year = 2018
end_year = 2048
ntrials = 1000  # number of simulations to run
years = np.arange(start_year, end_year + 1)  # create array of years
amount = np.zeros((ntrials, len(years)))  # empty array of amts
amount[0] = principal  # initial investment of $3000

# generate a list of random gaussian interest rates
mu, sigma = 0.08, 0.15 # mean and standard deviation
np.random.seed(41)
interest = np.random.normal(mu, sigma, size=(ntrials,len(years)))
```

In [15]:
```
# calculate the returns for n trials over the range of years
for i in range(ntrials):
    for year in range(len(years) - 1):
        amount[i,year + 1] = amount[i,year] * (1 + interest[i,year]) + princi

fig, ax = plt.subplots(1,2, figsize=(15,5),sharey=True)

fig.suptitle(f"Simulated {ntrials} Invenstent Runs", fontsize=16)
fig.tight_layout()
for amt in range(ntrials):
    ax[0].plot(years,amount[amt,:], color="blue", alpha=0.05)
ax[0].set_xlabel("Year")
ax[0].set_ylabel("\$\$")
ax[1].hist(amount[:,-1],bins=30, orientation="horizontal", color="black", alpl
ax[1].annotate(f"mean = ${round(np.mean(amount[:,-1]))}\n \
   std = ${round(np.std(amount[:,-1]))}",(10,1.5e6))
ax[1].set_xlabel("Counts");
```



B. Suppose now that the retirement account contains three classes of investments: Canadian stocks, foreign stocks, and bonds. The yields on these three investments each vary randomly but with some correlation. Here is the yield information for each investment:

> Canadian stock = mean 8%, SD 15%
>
> foreign stock = mean 8%, SD 15%
>
> Canadian bonds = mean 5%, SD 7%

> correlation between Canadian and foreign stock = 0.50
>
> correlation between Canadian stock and bonds = 0.20
>
> correlation between foreign stock and bonds = 0.05

On January 1 of each year you put $1000 into each class of investment. Show the distribution of the total amount of money in your account on Dec 31, 2047. What are the mean and SD?

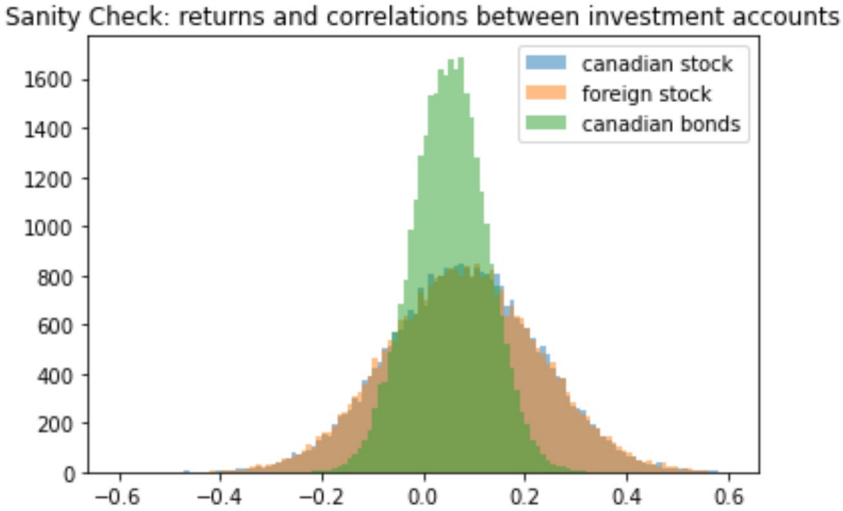
(Hint: generate a random yield drawn from a Gaussian distribution for the first investment. Then generate a second Gaussian random number, and form a linear combination of it and the first Gaussian number that will have the desired SD and covariance. This is the yield of the second investment. Then generate a third random number, and form a linear combination of all three that will give the correct SD and covariance for the yield of this combination with the other two yields.)

In [16]:

```
# generate 3 independent random yields from a gaussian distributions 1 39 2
mtrials = ntrials * len(years)

mu1, sigma1 = 0.08, 0.15
np.random.seed(12)
norm1 = np.random.normal(mu1, sigma1, size=(mtrials))

np.random.seed(39)
mu2, sigma2 = 0.08, 0.25
norm2 = np.random.normal(mu2, sigma2, size=(mtrials))

np.random.seed(222)
mu3, sigma3 = 0.05, 0.08
norm3 = np.random.normal(mu3, sigma3, size=(mtrials))

# create stocks using linear combinations of the three distributions
cadstock = norm1
forstock = (0.49 * cadstock + 0.51 * norm2 )# figure out the LC by guess and
cadbond = 0.115 * cadstock - 0.034 * forstock + 0.9 * norm3

# make sure i got this right
print(f"cadstock               mean: {np.mean(cadstock)}        stddev: {np.st
print(f"forstock               mean: {np.mean(forstock)}        stddev: {np.st
print(f"cadbond                mean: {np.mean(cadbond)}       stddev: {np.sto
print("")
print(f"corr cadstock forstock {np.corrcoef(cadstock, forstock)[0,1]}")
print(f"corr forstock cadbond  {np.corrcoef(forstock, cadbond)[0,1]}")
print(f"corr cadbond cadstock  {np.corrcoef(cadbond, cadstock)[0,1]}")

binwidth = 1e-2
plt.hist(forstock,  bins=np.arange(-0.6, 0.6 + binwidth, binwidth), alpha=0.5,
plt.hist(cadstock, bins=np.arange(-0.6, 0.6 + binwidth, binwidth), alpha=0.5,
plt.hist(cadbond, bins=np.arange(-0.6, 0.6 + binwidth, binwidth), alpha=0.5, l
plt.title("Sanity Check: returns and correlations between investment accounts'
plt.legend();
```

```
cadstock               mean: 0.07930104302465399        stddev: 0.149326210031
1883
```

```
forstock              mean: 0.07990874074845607        stddev: 0.147307924968
6795
cadbond               mean: 0.05156815272394065        stddev: 0.073495234593
1431

corr cadstock forstock 0.4958902452443705
corr forstock cadbond  0.05727606146212397
corr cadbond cadstock  0.2066553748133442
```

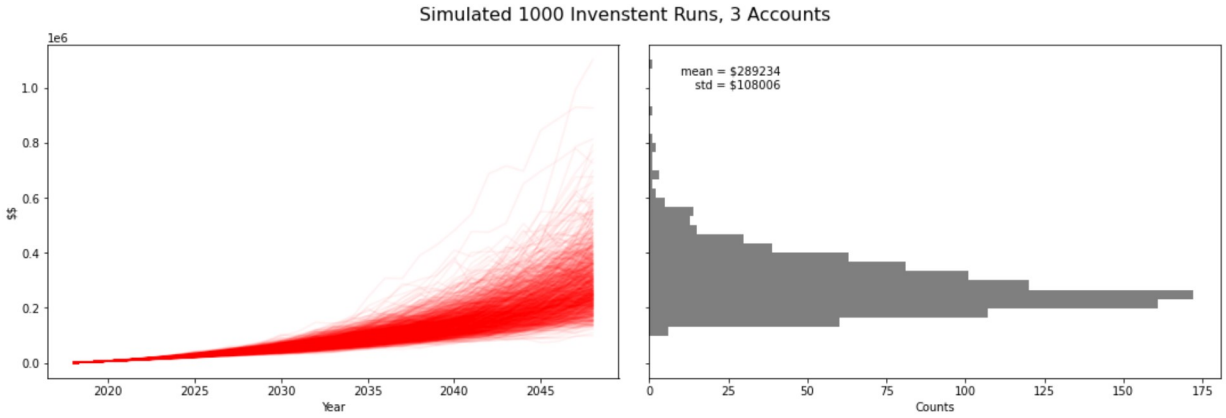


In [17]:

```
# initialize three accounts
principal = 1000
cs = np.zeros((ntrials, len(years)))  # canadian stocks
cs[0] = principal

fs = np.zeros((ntrials, len(years)))  # foreign stocks
fs[0] = principal

cb = np.zeros((ntrials, len(years)))  # canadian bonds
cb[0] = principal

# create a function that returns triplets
# canadian stock[i], foreign stock[i], canadian bonds[i]
# preserving the correlation between the random variables
# this could be made way more efficient... the point is
# we want a new set of random, but correlated interest
# rates for each year in each of our 1000 "trials", no repeating
def get_yield(j):
    return cadstock[j], forstock[j], cadbond[j]

# calculate each return
j = 0
for i in range(ntrials):
    for year in range(len(years) - 1):
        # get interest rates and increment
        cs_interest, fs_interest, cb_interest = get_yield(j)
        j += 1

        # generate interest in each account
        cs[i, year + 1] = cs[i, year] * (1 + cs_interest) + principal
        fs[i, year + 1] = fs[i, year] * (1 + fs_interest) + principal
        cb[i, year + 1] = cb[i, year] * (1 + cb_interest) + principal

total_money = cs + fs + cb
```

In [18]:

```
# do the plot
fig, ax = plt.subplots(1,2, figsize=(15,5),sharey=True)
fig.suptitle(f"Simulated {ntrials} Invenstent Runs, 3 Accounts", fontsize=16)
fig.tight_layout()
for amt in range(ntrials):
    ax[0].plot(years,total_money[amt,:], color="red", alpha=0.05)
ax[0].set_xlabel("Year")
ax[0].set_ylabel("\$\$")
ax[1].hist(total_money[:,-1],bins=30, orientation="horizontal", color="black",
ax[1].annotate(f"mean = ${round(np.mean(total_money[:,-1]))}\n \
   std = ${round(np.std(total_money[:,-1]))}",(10,1e6))
ax[1].set_xlabel("Counts");
```



C. Now suppose we add a procedure called "rebalancing". On January 1 of each year we contribute a total of $3000 to the account, but at the same time we redistribute the total amount of money in the account evenly between the three investments. How does this change the total amount on Dec 31, 2047? Show a plot of the distribution, and report the mean and SD as well.

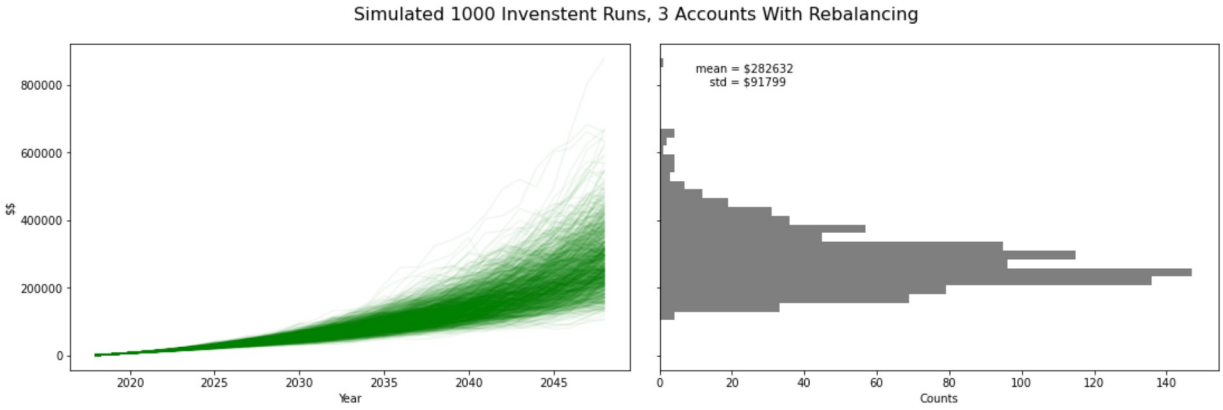
Commentary: You should find that although the total amount of money with rebalancing is slightly lower than in Part B, the SD is significantly smaller. The rebalancing procedure effectively forces you to sell investments when they're high and buy more of an investment when it's low. This effect is even more important when one accounts for the fact that the yield in one year is not completely independent of the yield in the next year---there is a small correlation that decreases with time which is not modelled in this problem. The net effect of this correlation is often that periods of high yield tend to be followed by periods of lower yield, and vice versa.

Retirement planning is in general a matter of optimizing the allocations to multiple types of investments in such a way as to yield the maximum probability of having "enough". High yield is of course good, but so is low variance. Having twice as much money as you need is a very different beast than having half as much as you need!

Ideally what you would want would be high yield investments with as little correlation as possible, so as to lower the variance. An investment strategy only slightly modified from the basic one in this problem, implemented with passively managed index funds, will generally outperform most actively management portfolios!

In [19]:
```
# same code as (B), just add rebalancing inside the loop

# initialize three accounts
principal = 1000
cs = np.zeros((ntrials, len(years)))  # canadian stocks
cs[0] = principal

fs = np.zeros((ntrials, len(years)))  # foreign stocks
fs[0] = principal

cb = np.zeros((ntrials, len(years)))  # canadian bonds
cb[0] = principal

# calculate each return
j = 0
for i in range(ntrials):
    for year in range(len(years) - 1):
        # get interest rates and increment
        cs_interest, fs_interest, cb_interest = get_yield(j)
        j += 1

        ##########################################################
        # do the rebalancing
        total_money = cs[i, year] + fs[i, year] + cb[i, year]
        cs[i, year] = total_money / 3
        fs[i, year] = total_money / 3
        cb[i, year] = total_money / 3
        ##########################################################

        # generate interest in each account
        cs[i, year + 1] = cs[i, year] * (1 + cs_interest) + principal
        fs[i, year + 1] = fs[i, year] * (1 + fs_interest) + principal
        cb[i, year + 1] = cb[i, year] * (1 + cb_interest) + principal

total_money = cs + fs + cb
```

In [20]:
```
# do the plot
fig, ax = plt.subplots(1,2, figsize=(15,5),sharey=True)

fig.suptitle(f"Simulated {ntrials} Invenstent Runs, 3 Accounts With Rebalanci
fig.tight_layout()
for amt in range(ntrials):
    ax[0].plot(years,total_money[amt,:], color="green", alpha=0.05)
ax[0].set_xlabel("Year")
ax[0].set_ylabel("\$\$")
ax[1].hist(total_money[:,-1],bins=30, orientation="horizontal", color="black",
ax[1].annotate(f"mean = ${round(np.mean(total_money[:,-1]))}\n \
   std = ${round(np.std(total_money[:,-1]))}",(10,8e5))
ax[1].set_xlabel("Counts");
```


Simulated 1000 Invenstent Runs, 3 Accounts With Rebalancing
mean = $282632
std = $91799
$$
Year
Counts
0
200000
400000
600000
800000
2020
2025
2030
2035
2040
2045
0
20
40
60
80
100
120
140