Monte Carlo Simulation
for
A Simple Portfolio Problem

Ken Judd
Penn State University
March, 2019

In[3606]:=
```
a = 0; Remove["Global`*"]
```

In this notebook, I will show that Monte Carlo integration does not do a good job in solving a simple portfolio problem.

The basic problem is that an enormous number of draws is necessary to get good accuracy when you take the conventional approach used in economics.

This problem is revealed by simple experiments where we solve the problem several times using different seeds for our random number generator.

I will then illustrate the surrogate function approach where we first use a moderate number of draws to create a good approximation (called a surrogate) of the objective function, and then optimize the surrogate function.

Closed-Form solution

Description

Portfolio problem:
one safe asset ($1 invested today produces $R tomorrow)
one risky asset ($1 invested today produces $Z tomorrow, Z is Normal[$\mu$,$\sigma$])
exponential utility function (constant absolute risk aversion, ARA)

General case

Constant absolute risk aversion utility

In[3607]:=
```
util[x_] = -Exp[-ARA x];
```

If you have one unit of wealth and put a fraction θ in the risky asset then your wealth tomorrow will be the random variable W:

In[3608]:=
```
W = θ Z + (1 - θ) R;
```

In[3609]:=
```
dist = NormalDistribution[μ, σ];
TrueMoments = {Mean[dist], StandardDeviation[dist], Skewness[dist], Kurtosis[dist]}
```

Out[3610]=
$$\{\mu, \sigma, 0, 3\}$$

Compute expected utility as a function of theta

In[3611]:=
```
EU[θ_] = Expectation[util[W], Z ≈ NormalDistribution[μ, σ]]
```

Out[3611]=
$$-e^{\text{ARA R } (-1+\theta)+\frac{1}{2}\text{ ARA } \theta \left(-2\,\mu+\text{ARA } \theta\, \sigma^2\right)}$$

Solve for the optimal theta

In[3612]:=
```
EU'[θ] // FullSimplify
```

Out[3612]=
$$-\text{ARA } e^{\text{ARA R } (-1+\theta)+\frac{1}{2}\text{ ARA } \theta \left(-2\,\mu+\text{ARA } \theta\, \sigma^2\right)} \left(\text{R}-\mu+\text{ARA } \theta\, \sigma^2\right)$$

The first terms of this expression are never zero, so we find the zero of the last term.

In[3613]:=
```
Solve[(% // Last) == 0, θ][[1]]
```

Out[3613]=
$$\left\{\theta \to \frac{-\text{R}+\mu}{\text{ARA } \sigma^2}\right\}$$

In[3614]:=
```
θsol = θ /. %
```

Out[3614]=
$$\frac{-\text{R}+\mu}{\text{ARA } \sigma^2}$$

Specific case

For the remainder of this notebook, we examine one specific example defined by the following parameter choices:

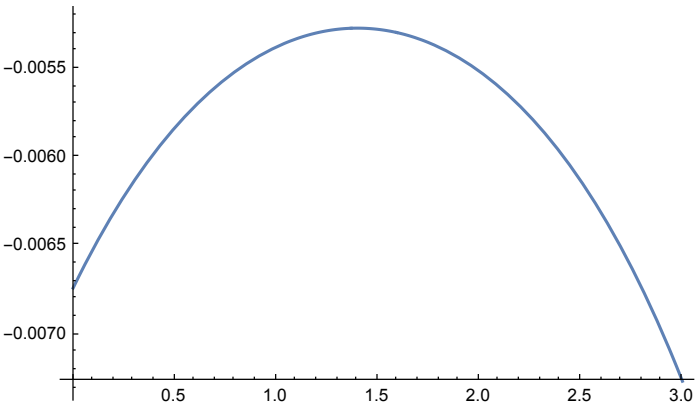
In[3615]:=
```
R = 1; μ = 1.07; σ = 1 / 10.; ARA = 5;
```

The expected utility function is a function of θ.

In[3616]:=
```
EU[x]
Plot[EU[x], {x, 0, 3}]
```

Out[3616]=
$$-e^{5\,(-1+x)+\frac{5}{2}\,(-2.14+0.05\,x)\,x}$$

Out[3617]=



The computational challenge

This example is function EU[θ] is usually difficult to compute because there is no analytic solution to the integration. The common approach to this is to use Monte Carlo methods to approximate the integral.

We will show that there are two ways to use MC methods to solve this problem - the “common random variables” method and the “surrogate function” approach.

Standard approach for applying simulation

In[3618]:=
```
SetOptions[FindMaximum, {AccuracyGoal → 6, PrecisionGoal → 6}];
SetOptions[ListPlot, PlotStyle → {Red, PointSize[Large]}];
```

The standard procedure consists of three steps

1: Make one set of random draws for Z,

2: In the definition of EU[θ], replace the random variable Z with those draws, creating a function EUSim[θ] which we use as an approximation of EU[θ]

3: Use EUSim[θ] in the optimization problem

Step-by-step description with Ndraws = 10

We first do a very simple example where you can see all the details.

```
(* Make 10 draws *)
Ndraws = 10;
returns = RandomReal[NormalDistribution[μ, σ], Ndraws] // Sort
```

Out[3621]= {0.798023, 0.988623, 1.00358, 1.01451, 1.01556, 1.05103, 1.10515, 1.124, 1.20967, 1.36337}

```
(* Compute the vector of wealths *)
W = θ returns + (1 - θ) R
```

Out[3622]= {1 − 0.201977 θ, 1 − 0.0113771 θ, 1 + 0.00358134 θ, 1 + 0.0145133 θ, 1 + 0.0155587 θ, 1 + 0.0510266 θ, 1 + 0.105149 θ, 1 + 0.124004 θ, 1 + 0.209671 θ, 1 + 0.363375 θ}

```
(* Define expected utility as function of θ *)
EUSim[θ_] = Mean@util@W
```

Out[3623]= $\frac{1}{10}\left(-e^{-5\,(1-0.201977\,\theta)} - e^{-5\,(1-0.0113771\,\theta)} - e^{-5\,(1+0.00358134\,\theta)} - e^{-5\,(1+0.0145133\,\theta)} - e^{-5\,(1+0.0155587\,\theta)} - e^{-5\,(1+0.0510266\,\theta)} - e^{-5\,(1+0.105149\,\theta)} - e^{-5\,(1+0.124004\,\theta)} - e^{-5\,(1+0.209671\,\theta)} - e^{-5\,(1+0.363375\,\theta)}\right)$

```
(* Compute optimal portfolio weight *)
th = θ /. FindMaximum[EUSim[θ], {θ, 0., 3.}][[2]];
(* Display optimal portfolio and estimated utility *)
{th, EUSim[th]}
```

Out[3625]= {0.711054, −0.00599001}

Analysis of error

EUSim is an approximation of EU. When we plot EUSim and EU, we see that EUSim is not a good approximation of EU. This is not surprising when we use so few draws, but let us examine the nature of the error

Compare the moments of the true distribution and the 10 draws we made.

```
dist = returns;
{{"mean", "standard deviation", "skewness", "kurtosis"},
  TrueMoments, {Mean[dist], StandardDeviation[dist],
   Skewness[dist], Kurtosis[dist]}} // TableForm
```

Out[3658]//TableForm=

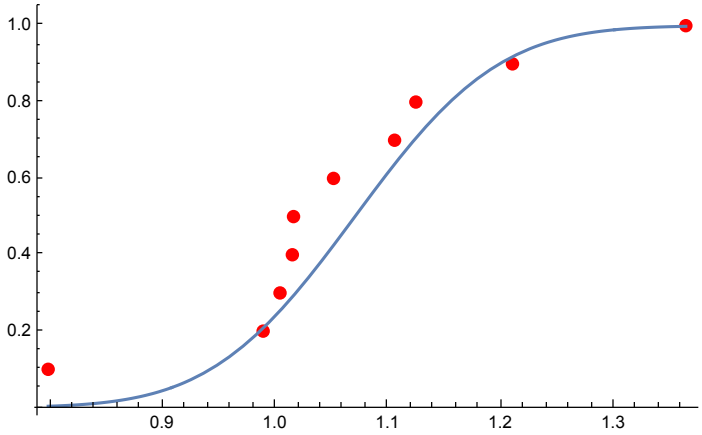
mean	standard deviation	skewness	kurtosis
1.07	0.1	0	3
1.06735	0.149468	0.297561	3.3228

The moments of the draws differ significantly from the true moments. We see this when we plot the

CDF of the true distribution along with the CDF of the 10 draws.

In[3667]:=
```
(* Plot the CDF *)
gr1 = ListPlot[{returns // Sort, Range[1, Ndraws] / 10} // Transpose];
min = Min[returns]; max = Max[returns];
gr2 = Plot[CDF[NormalDistribution[μ, σ], x] // Evaluate, {x, min, max}];
Show[gr1, gr2]
```

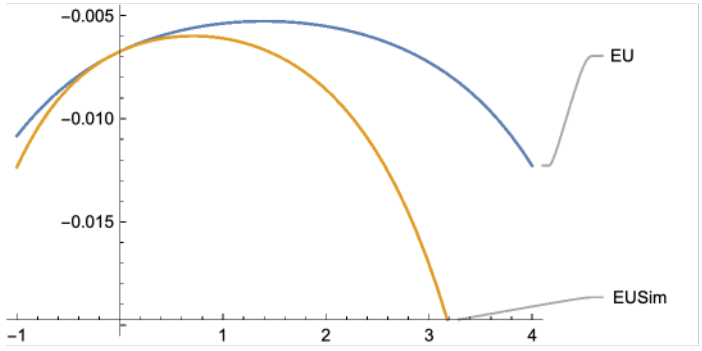
Out[3670]=



Plot EU and the approximation EUSim. We find that EUSim is generally below EU. This is consistent with the simulated standard deviation being greater than the true standard deviation

In[3671]:=
```
Plot[{EU[θ], EUSim[θ]}, {θ, -1, 4}, PlotLabels → {"EU", "EUSim"}]
```

Out[3671]=



The key fact is that we use the same draws each time we compute EU[θ], which means that the standard deviation is too high at each θ.

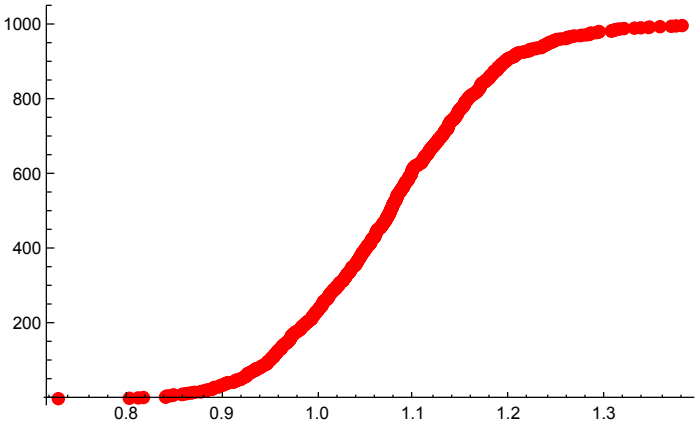
More serious applications

Ndraws=1000

The Ndraws=5 example did not do well because the number of draws was so small. We now do a serious example.

In[3206]:=
```
Ndraws = 1000;
(* Draw returns *)
returns = RandomReal[NormalDistribution[μ, σ], Ndraws];
(* Plot the CDF *)
ListPlot[{returns // Sort, Range[1, Ndraws]} // Transpose]
```

Out[3208]=

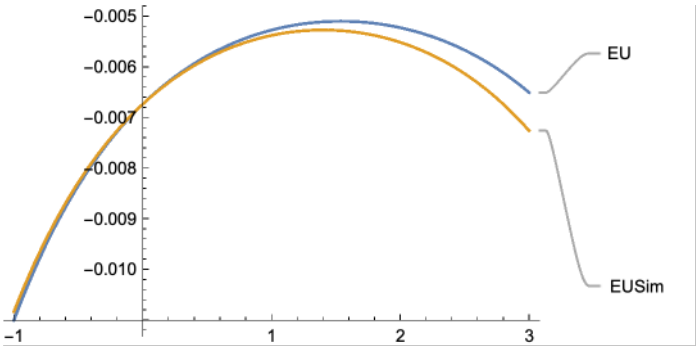


In[3209]:=
```
(* Compute ex post W over those draws as a function of theta *)
W = θ returns + (1 - θ) R;
(* Define expected utility as function of θ *)
EUSim[θ_] = Mean@util@W;
```

Plot EU and EUSim. We see that EUSim is a moderately good approximation of EU

In[3211]:=
```
Plot[{EUSim[θ], EU[θ]}, {θ, -1, 3}, PlotLabels → {"EU", "EUSim"}]
```

Out[3211]=



In[3212]:=
```
(* Compute optimal portfolio weight *)
th = θ /. FindMaximum[EUSim[θ], {θ, 0., 3.}][[2]];
(* Display optimal portfolio and estimated utility *)
{th, EUSim[th]}
```

Out[3213]= {1.53752, −0.00510098}

The estimate is not a good one.

Ndraws=1000 does better. However, we need to remember that the solution to the optimization problem is a random variable because it depends on the random vector of draws. We want to determine the variance of the solution. We do this by repeating the procedure Ntests times, each time using a different set of draws for the returns.

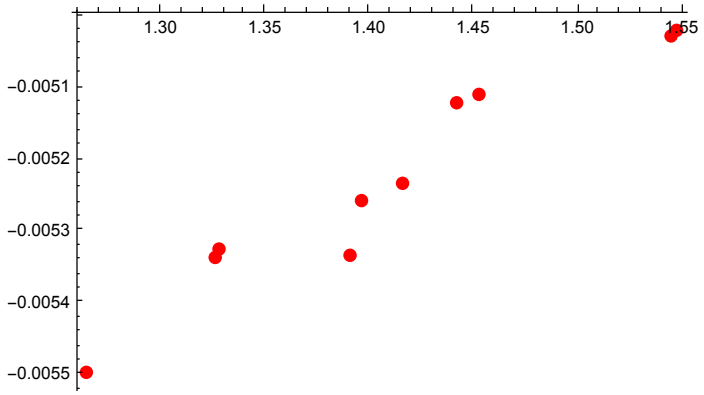
The following does this Ntests times.

In[3318]:=
```
resultss := (
  results = Table[
    returns = RandomReal[NormalDistribution[μ, σ], Ndraws];
    W = θ returns + (1 - θ) R;
    Clear[EUSim]; EUSim[θ_] = Mean@util@W;
    th = θ /. FindMaximum[EUSim[θ], {θ, 6., 8.}][[2]];
    {th, EUSim[th]},
    {Ntests}];
  ListPlot[results] // Print;
  resultsT = results // Transpose;
  Print["mean solution = ", Mean[resultsT[[1]]],
   "    stdev = ", StandardDeviation[resultsT[[1]]]]
 )
```

Set Ntests to 10

In[3319]:=
```
Ntests = 10;
```

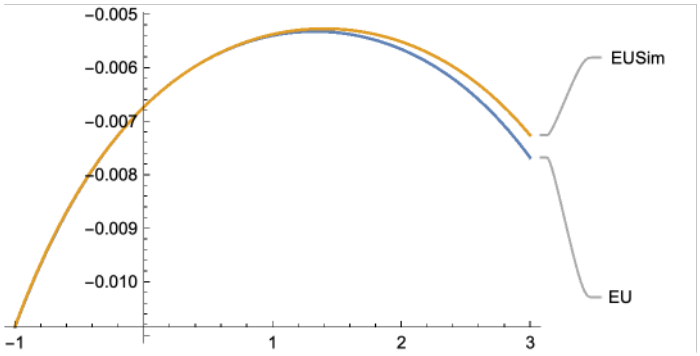
In[3320]:=
```
Ndraws = 1000;
time = resultss // AbsoluteTiming // First; Print ["time = ", time]
```




```
mean solution = 1.41042    stdev = 0.0915779
time = 0.345418
```

The average estimate is good, but the individual ones are spread out over a nontrivial interval. The plot below looks at the approximation in the last test.

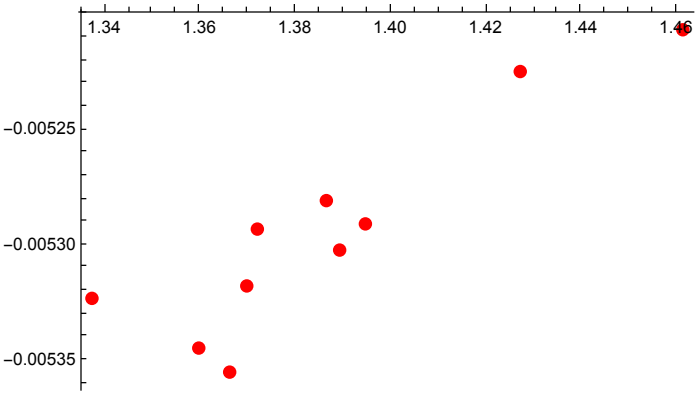
```
Plot[{EUSim[θ] // Evaluate, EU[θ]}, {θ, -1, 3}, PlotLabels → {"EU", "EUSim"}]
```



Ndraws=10,000

Repeat but with Ndraws=10,000

In[3322]:=
```
Ndraws = 10 000;
time = resultss // AbsoluteTiming // First; Print ["time = ", time]
```

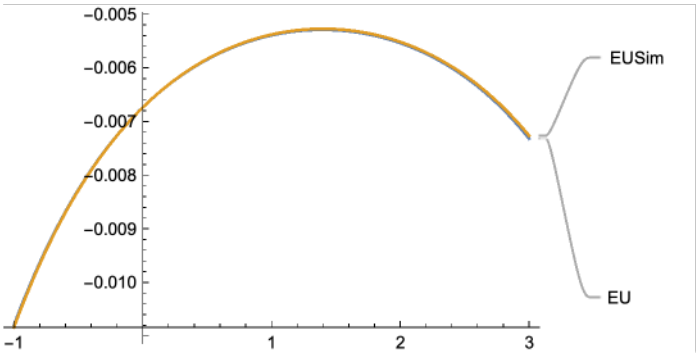



```
mean solution = 1.38628     stdev = 0.035513

time = 3.83394
```

In[3324]:=
```
Plot[{EUSim[θ] // Evaluate, EU[θ]}, {θ, -1, 3}, PlotLabels → {"EU", "EUSim"}]
```

Out[3324]=

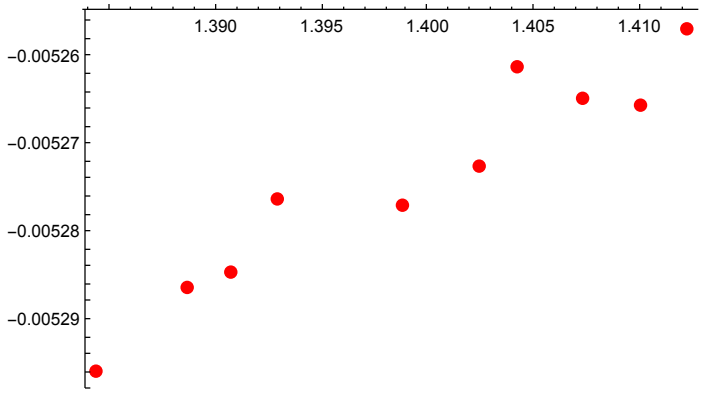


The average solution is very close to the true solution, but the solutions were still scattered over a montrivial range and the standard deviation was not negligible.

Ndraws=100,000

Repeat but with Ndraws=100,000

In[3218]:=
```
Ndraws = 100 000;
time = resultss // AbsoluteTiming // First; Print ["time = ", time]
```




```
mean solution = 1.39912     stdev = 0.00962624

time = 47.3467
```

This is better but the time has gone up in proportion to Ndraws.

Surrogate Function Method: Approximating the objective

In[3351]:=
```
Chebpts[xmin_, xmax_, npts_] :=
 ((xmin + xmax) / 2 + Cos[(2 Range[1, npts] - 1) π / (2 npts)] (xmax - xmin) / 2) // Reverse
```

Idea

We need to approximate EU[θ] in some way. EUSim[θ] is one way to do this. It replaces the random variable with a finite set of realizations and uses those realizations for each value of θ.

The surrogate function method uses approximation theory to construct an approximation. Approximation theory first approximates EU[θ] at each θ in some prespecified set, and then uses some curve-fitting method (regression, splines, etc.) to construct an approximation.

Strategy

Pick a finite set of thetas, thetas = {θ_1, θ_1, θ_1 ... θ_{Nthetas}}
At each θ_i, use Monte Carlo integration to approximate EU[θ_i]
Use regression to approximate the EU[θ] function; call that EUSurr[θ]
Maximize EUSurr[θ]

Define some commands

Every time we invoke ExpUtil it will cause returns to be evaluated, resulting in a new set of draws, and then the state-contingent W vector is computed, and then the expected utility is computed.

In[3352]:=
```
returns := RandomReal[NormalDistribution[μ, σ], Ndraws];
W := theta returns + (1 - theta) R;
ExpUtil := Mean@util@W;
```

Step-by-step description

In[3355]:=
```
(* Choose range and number of thetas (Nthetas) to be examined *)
thmin = -1; thmax = 4.; thmean = (thmin + thmax) / 2;
Nthetas = 5; θvals = Chebpts[thmin, thmax, Nthetas]
```

Out[3356]=
```
{-0.877641, 0.0305369, 1.5, 2.96946, 3.87764}
```

In[3357]:=
```
(* Choose number of draws *)
Ndraws = 1 000 000;
```

Compute the expected utility for each θ, using a different set of 1,000,000 Normal draws for each θ

In[3358]:=
```
EUvals = Range[1, Nthetas]; (* create a vector for expected utilities *);
```

In[3359]:=
```
Do[
 theta = θvals[[i]];
 EUvals[[i]] = ExpUtil,
 {i, 1, Nthetas}]
```

Collect data.

In[3364]:=
```
data = {θvals, EUvals} // Transpose;
% // TableForm
```

Out[3365]//TableForm=

-0.877641	-0.0100871
0.0305369	-0.00666706
1.5	-0.00528168
2.96946	-0.00719419
3.87764	-0.0113286

We next define the functional form that we will use in a regression.
model will be the functional form with variables a, b, and c.

In[3366]:=
```
Clear[model, a, b, c];
model = a + b θ + c θ^2;
vars = {a, b, c};
```

The next command computes a least-squares fit of the data to the model

In[3369]:=
```
fit = FindFit[data, model, vars, θ]
```

Out[3369]=
```
{a → -0.00688479, b → 0.00274157, c → -0.000993355}
```

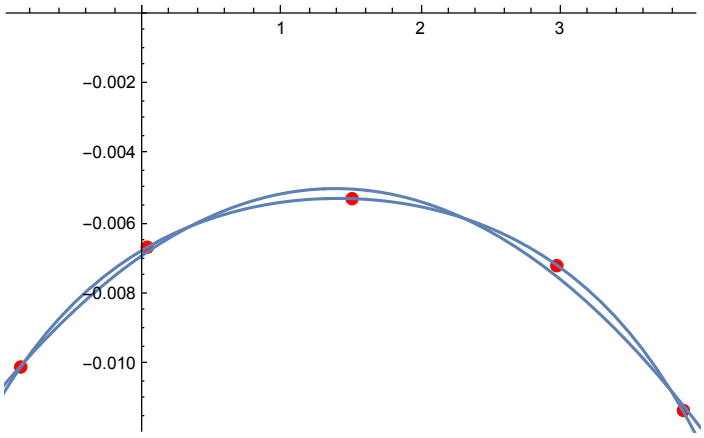
We use these regression coefficients to define value

In[3370]:=
```
EUSurr[θ_] = model /. fit;
```

```
gr0 = Plot[EU[θ], {θ, thmin, thmax}];
```

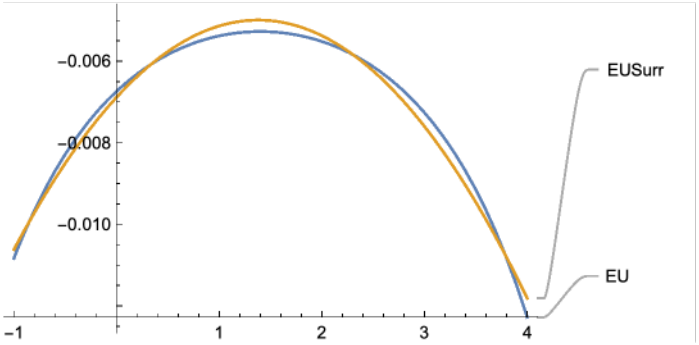
In[3372]:=
```
gr1 = ListPlot[data]; gr2 = Plot[EUSurr[θ], {θ, thmin, thmax}];
Show[gr1, gr2, gr0]
```

Out[3373]=



In[3374]:=
```
Plot[{EU[θ], EUSurr[θ]}, {θ, thmin, thmax}, PlotLabels → {"EU", "EUSurr"}]
```

Out[3374]=



Find optimal θ

In[3375]:=
```
thsol = θ /. FindMaximum[{EUSurr[θ], {thmin ≤ θ ≤ thmax}}, {θ, thmean}][[2]]
```

Out[3375]=
```
1.37996
```

The next code
1) sets Nthetas, θvals, and Ndraws
2) computes expected utility for each θ in θvals,
3) computes a curve that runs through those points, and
4) finds the max of the regression curve.

In[3376]:=
```
run :=
 (EUvals = θvals;
  Do[
   theta = θvals[[itheta]]; EUvals[[itheta]] = Mean@util@W,
   {itheta, 1, Nthetas}];
  data = {θvals, EUvals} // Transpose;
  gr1 = ListPlot[data];
  fit = FindFit[data, model, vars, θ];
  Clear[EUSurr]; EUSurr[θ_] = model /. fit;
  gr2 = Plot[EUSurr[theta], {theta, thmin, thmax}];
  thsol = θ /. FindMaximum[{EUSurr[θ], {thmin ≤ θ ≤ thmax}}, {θ, thmean}][[2]]
 )
```

We define a script that will call run Ncases times and report the results

In[3377]:=
```
rrun := (start = AbsoluteTime[];
  results = Table[run, {Ncases}];
  time = AbsoluteTime[] - start;
  Print[ "Time for ", Ncases, "  cases was ", time, " seconds"];
  Print["mean estimate is  ", Mean[results],
   "  with standard deviation  ", StandardDeviation[results]])
```

In[3378]:=
```
Clear[model, a, b, c];
 model = a + b θ + c θ^2;
vars = {a, b, c};
```

In[3381]:=
```
thmin = -1; thmax = 4.; thmean = (thmin + thmax) / 2; Nthetas = 11;
θvals = Chebpts[thmin, thmax, Nthetas]
```

Out[3382]=
```
{-0.974554, -0.77408, -0.389374, 0.148398,
 0.795669, 1.5, 2.20433, 2.8516, 3.38937, 3.77408, 3.97455}
```

In[3383]:=
```
Ndraws = 10 000; Ncases = 10;
```

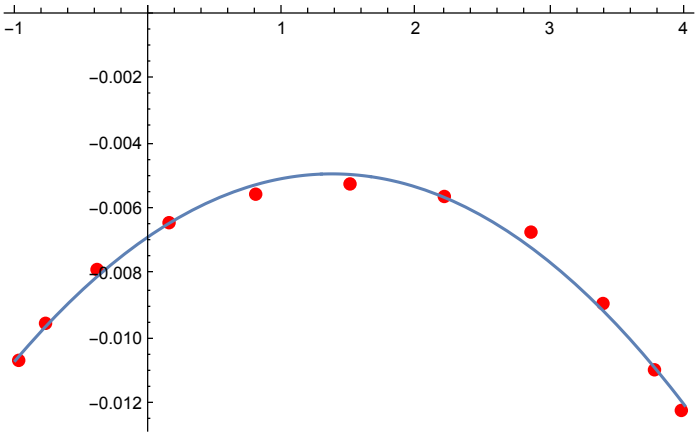
In[3384]:=
```
rrun
```

```
Time for 10  cases was 0.419052 seconds
```

```
mean estimate is  1.38758  with standard deviation  0.0207673
```

In[3385]:=
```
Show[gr1, gr2]
```

Out[3385]=



In[3386]:=
```
Ndraws = 100 000; Ncases = 10;
```

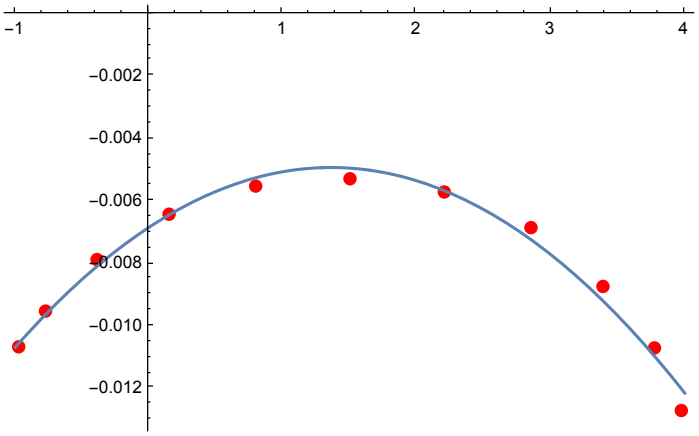
In[3387]:=
```
rrun
```

```
Time for 10  cases was 0.657793 seconds
```

```
mean estimate is  1.38135  with standard deviation  0.0112333
```

In[3388]:=
```
Show[gr1, gr2]
```

Out[3388]=



In[3389]:=
```
Ndraws = 1 000 000; Ncases = 10;
```

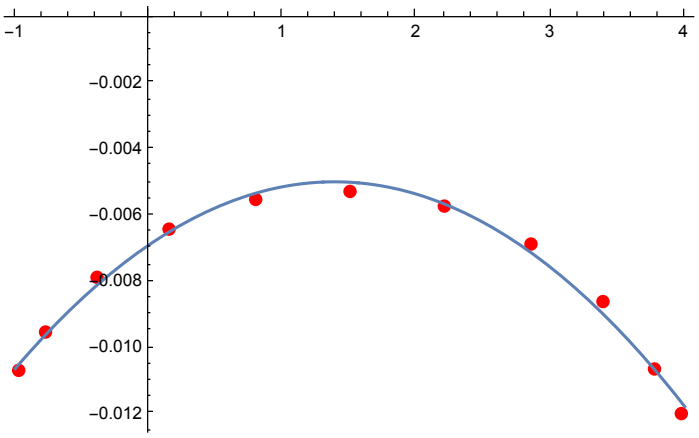
In[3390]:=
```
rrun
```

```
Time for 10  cases was 3.569983 seconds

mean estimate is  1.38006  with standard deviation  0.00274248
```

In[3391]:=
```
Show[gr1, gr2]
```

Out[3391]=



second iteration: reduce interval

The solution in the first iteration was 1.38. The approximation was a quadratic function over a wide range. We next repeat the procedure but now over a smaller range that contains the first solution. In this case, we choose the interval [1, 2].

In[3392]:=
```
thmin = 1.; thmax = 2.; thmean = (thmin + thmax) / 2;
θvals = Chebpts[thmin, thmax, Nthetas]
```

Out[3393]=
```
{1.00509, 1.04518, 1.12213, 1.22968, 1.35913,
 1.5, 1.64087, 1.77032, 1.87787, 1.95482, 1.99491}
```

We now approximate the objective function over this smaller range for a total of 10 times, each time with a different seed:

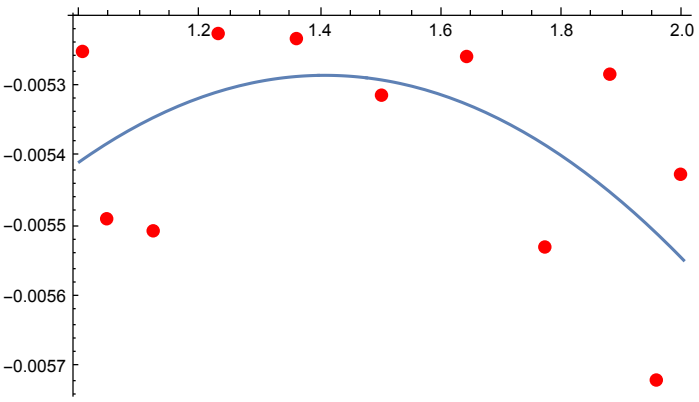
In[3394]:=
```
Ndraws = 1000; rrun
```

```
Time for 10  cases was 0.399605 seconds
```

```
mean estimate is  1.33282  with standard deviation  0.144059
```

In[3395]:=
```
Show[gr1, gr2]
```

Out[3395]=



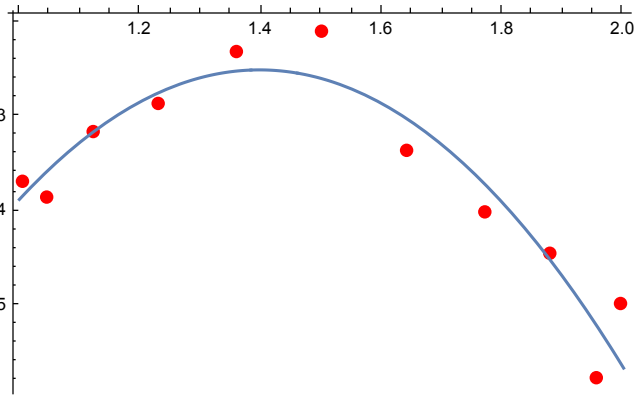
In[3396]:=
```
Ndraws = 10 000; rrun
```

```
Time for 10  cases was 0.409211 seconds
```

```
mean estimate is  1.40411  with standard deviation  0.0259402
```

In[3397]:=
```
Show[gr1, gr2]
```

Out[3397]=



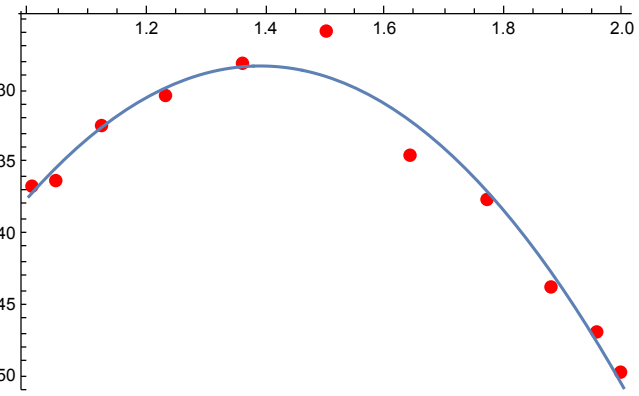
In[3398]:=
```
Ndraws = 100 000; rrun
```

```
Time for 10  cases was 0.650467 seconds
```

```
mean estimate is  1.39976  with standard deviation  0.00996922
```

In[3400]:=
```
Show[gr1, gr2]
```

Out[3400]=



Very good solutions with small variation

Quartic model, Nthetas=21

This time we use a degree 4 polynomial approximation

In[3401]:=
```
Clear[model]; model = a + b θ + c θ^2 + d θ^3 + e θ^4; vars = {a, b, c, d, e};
```

In[3402]:=
```
thmin = 0.; thmax = 4.; thmean = (thmin + thmax) / 2;
```

In[3403]:=
```
Nthetas = 21;
θvals = Chebpts[thmin, thmax, Nthetas]
```

Out[3404]=
```
{0.00559241, 0.0501442, 0.138253, 0.267949, 0.436337, 0.639655,
 0.87336, 1.13223, 1.41049, 1.70192, 2., 2.29808, 2.58951, 2.86777,
 3.12664, 3.36035, 3.56366, 3.73205, 3.86175, 3.94986, 3.99441}
```

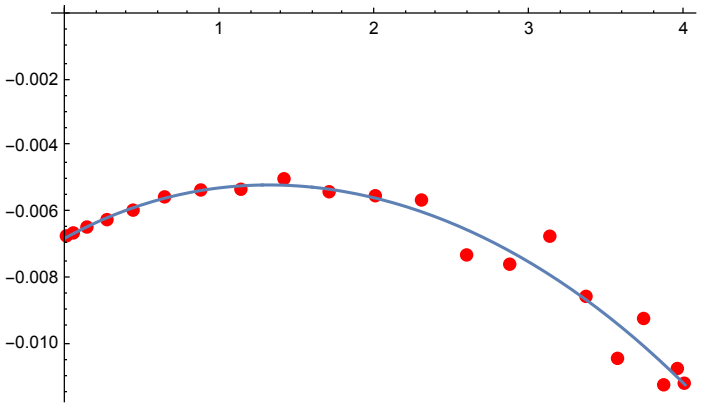
In[3405]:=
```
Ndraws = 1000; rrun
```

```
Time for 10  cases was 0.460238 seconds
```

```
mean estimate is  1.29856  with standard deviation  0.247833
```

In[3406]:=
```
Show[gr1, gr2]
```

Out[3406]=



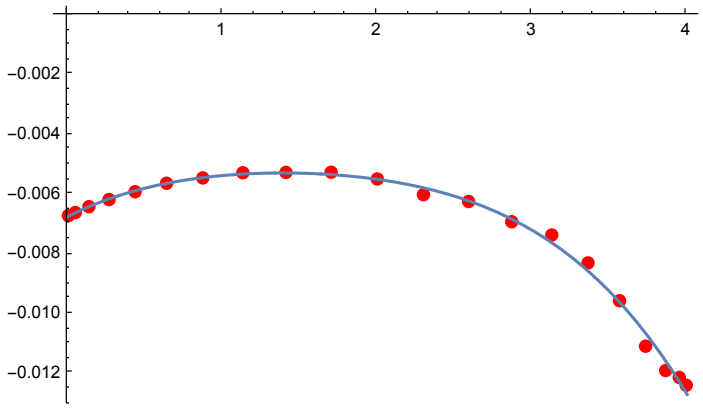
In[3407]:=
```
Ndraws = 10 000; rrun
```

```
Time for 10  cases was 0.463709 seconds
```

```
mean estimate is  1.3838  with standard deviation  0.0786548
```

In[3408]:=
```
Show[gr1, gr2]
```

Out[3408]=



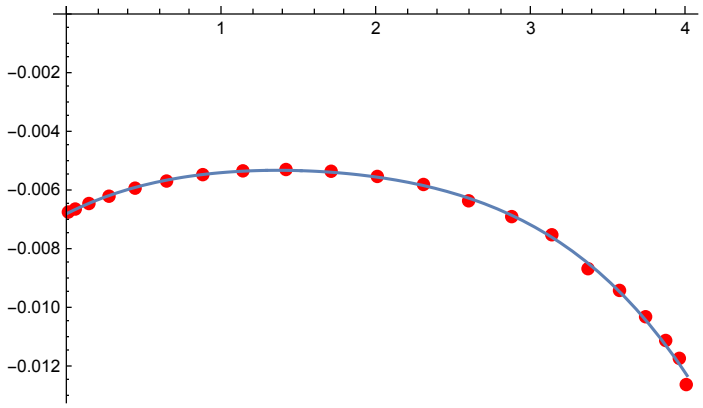
In[3409]:=
```
Ndraws = 100 000; rrun
```

```
Time for 10  cases was 0.912983 seconds
```

```
mean estimate is  1.36959  with standard deviation  0.0376642
```

In[3410]:=
```
Show[gr1, gr2]
```

Out[3410]=



In[3411]:=
```
thmin = 1.; thmax = 2; thmean = (thmin + thmax) / 2;
```

In[3412]:=
```
Nthetas = 21;
θvals = Chebpts[thmin, thmax, Nthetas]
```

Out[3413]=
```
{1.0014, 1.01254, 1.03456, 1.06699, 1.10908, 1.15991,
 1.21834, 1.28306, 1.35262, 1.42548, 1.5, 1.57452, 1.64738, 1.71694,
 1.78166, 1.84009, 1.89092, 1.93301, 1.96544, 1.98746, 1.9986}
```

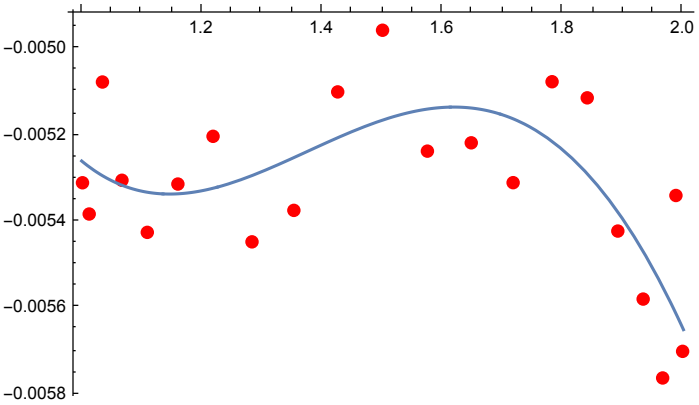
In[3414]:=
```
Ndraws = 1000; rrun
```

```
Time for 10  cases was 0.417582 seconds
```

```
mean estimate is  1.42975  with standard deviation  0.190921
```

In[3415]:=
```
Show[gr1, gr2]
```

Out[3415]=



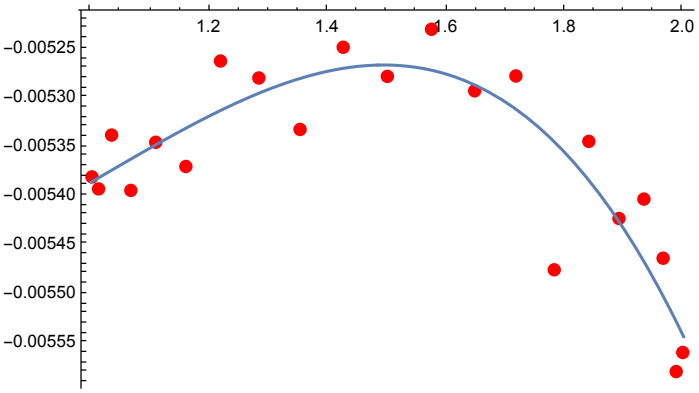
In[3416]:=
```
Ndraws = 10 000; rrun
```

```
Time for 10  cases was 0.468097 seconds

mean estimate is  1.40729  with standard deviation  0.0906905
```

In[3417]:=
```
Show[gr1, gr2]
```

Out[3417]=



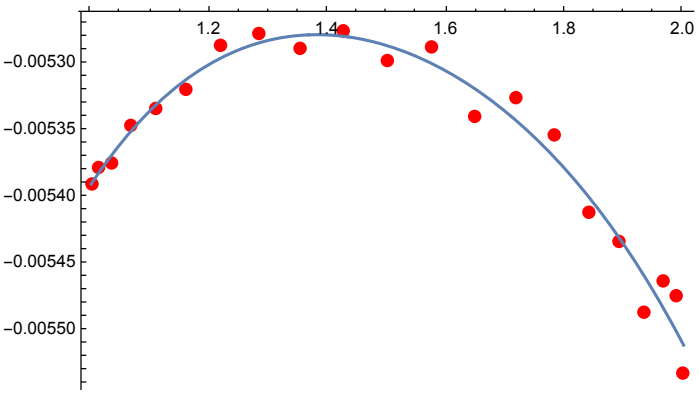
In[3418]:=
```
Ndraws = 100 000; rrun
```

```
Time for 10  cases was 0.927141 seconds

mean estimate is  1.3995  with standard deviation  0.0210758
```

In[3419]:=
```
Show[gr1, gr2]
```

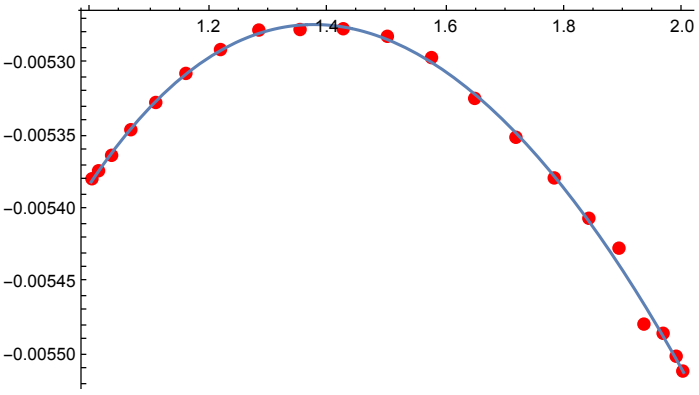
Out[3419]=



In[3420]:=
```
Ndraws = 1 000 000; rrun
```
```
Time for 10  cases was 8.347595 seconds
mean estimate is  1.39621  with standard deviation  0.0101574
```

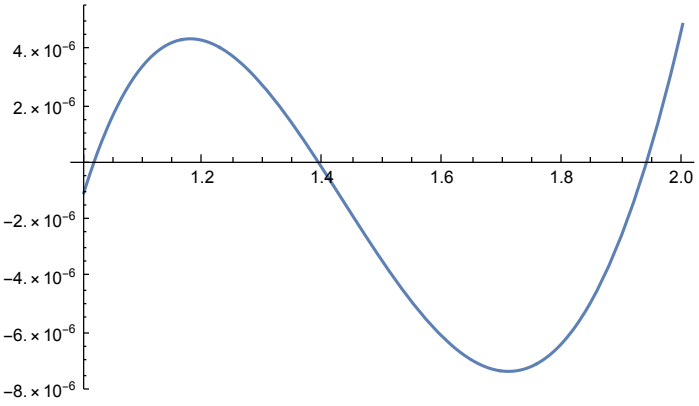
In[3421]:=
```
Show[gr1, gr2]
```

Out[3421]=



In[3422]:=
```
Plot[{EUSurr[θ] - EU[θ]}, {θ, thmin, thmax}]
```

Out[3422]=



In[3423]:=

Degree six polynomial model, Nthetas=21

This time we use a degree 6 polynomial approximation

In[3424]:=
```
Clear[model]; model = a + b θ + c θ^2 + d θ^3 + e θ^4 + f θ^5 + g θ^6; vars = {a, b, c, d, e, f, g};
```

In[3425]:=
```
thmin = 0.; thmax = 4.; thmean = (thmin + thmax) / 2;
```

In[3426]:=
```
Nthetas = 21;
θvals = Chebpts[thmin, thmax, Nthetas]
```

Out[3427]=
```
{0.00559241, 0.0501442, 0.138253, 0.267949, 0.436337, 0.639655,
 0.87336, 1.13223, 1.41049, 1.70192, 2., 2.29808, 2.58951, 2.86777,
 3.12664, 3.36035, 3.56366, 3.73205, 3.86175, 3.94986, 3.99441}
```

In[3428]:=
```
Ndraws = 1000; rrun
```

```
Time for 10  cases was 0.470788 seconds
mean estimate is  1.42231  with standard deviation  0.269263
```

In[3429]:=
```
Ndraws = 10 000; rrun
```

```
Time for 10  cases was 0.490968 seconds
mean estimate is  1.4105  with standard deviation  0.115562
```

In[3430]:=
```
Ndraws = 100 000; rrun
```

```
Time for 10  cases was 1.453138 seconds
mean estimate is  1.39276  with standard deviation  0.0429931
```

In[3431]:=
```
thmin = 1.; thmax = 2; thmean = (thmin + thmax) / 2;
```

In[3432]:=
```
Nthetas = 21;
θvals = Chebpts[thmin, thmax, Nthetas]
```

Out[3433]=
```
{1.0014, 1.01254, 1.03456, 1.06699, 1.10908, 1.15991,
 1.21834, 1.28306, 1.35262, 1.42548, 1.5, 1.57452, 1.64738, 1.71694,
 1.78166, 1.84009, 1.89092, 1.93301, 1.96544, 1.98746, 1.9986}
```

In[3434]:=
```
Ndraws = 1000; rrun
```

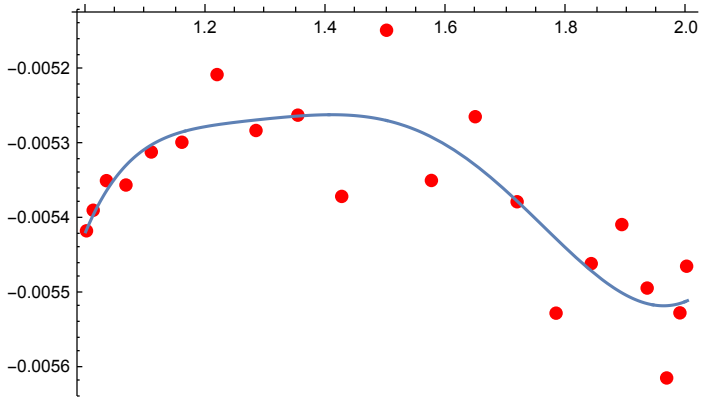
```
Time for 10  cases was 0.917441 seconds
mean estimate is  1.47976  with standard deviation  0.16556
```

In[3435]:=
```
Ndraws = 10 000; rrun
```

```
Time for 10  cases was 2.276581 seconds
mean estimate is  1.41588  with standard deviation  0.121692
```

In[3436]:=
```
Show[gr1, gr2]
```

Out[3436]=

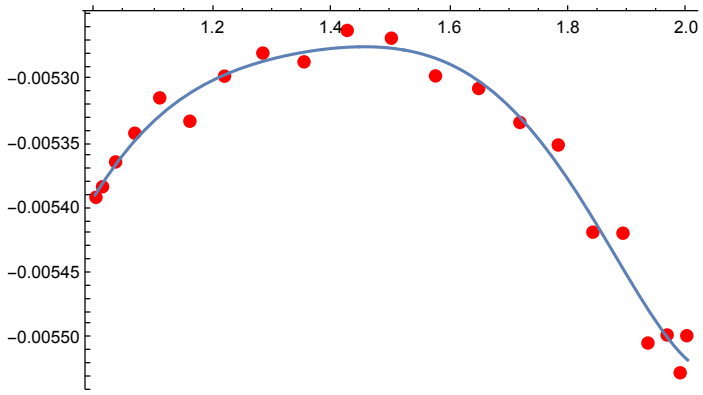


In[3437]:=
```
Ndraws = 100 000; rrun
```

```
Time for 10  cases was 0.907011 seconds
mean estimate is  1.413  with standard deviation  0.0425698
```

In[3438]:=
```
Show[gr1, gr2]
```

Out[3438]=

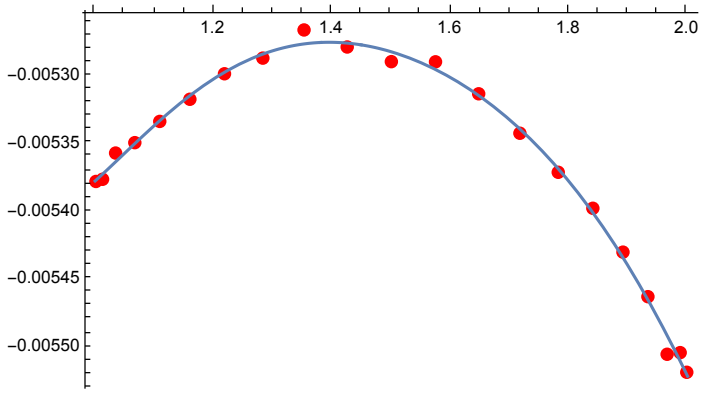


In[3439]:=
```
Ndraws = 1 000 000; rrun
```

```
Time for 10  cases was 7.215463 seconds
mean estimate is  1.3919  with standard deviation  0.00960508
```

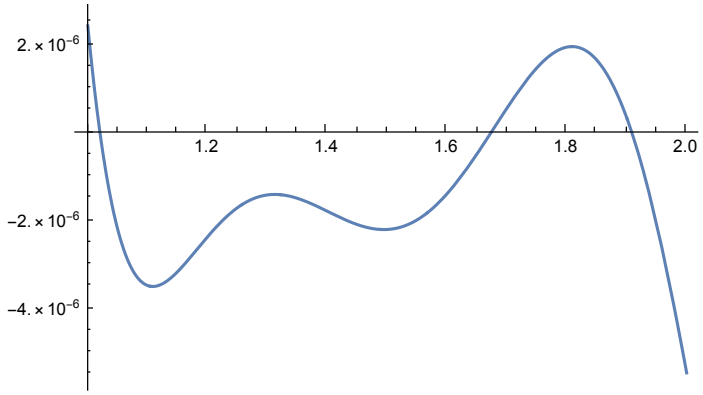
In[3440]:=
```
Show[gr1, gr2]
```

Out[3440]=



In[3441]:=
```
Plot[{EUSurr[θ] - EU[θ]}, {θ, thmin, thmax}]
```

Out[3441]=

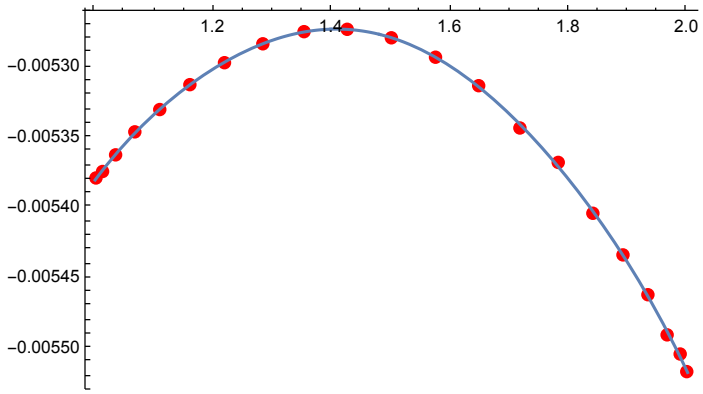


In[3442]:=
```
Ndraws = 10 000 000; rrun
```

```
Time for 10  cases was 65.430050 seconds
mean estimate is  1.40116  with standard deviation  0.00581504
```

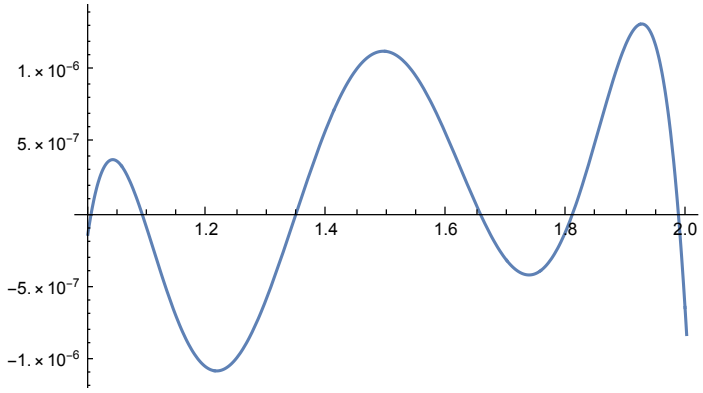
In[3443]:=
```
Show[gr1, gr2]
```

Out[3443]=



In[3444]:=
```
Plot[{EUSurr[θ] - EU[θ]}, {θ, thmin, thmax}]
```

Out[3444]=



In[3445]:=