VFI instability

In[54]:=
```
x = 0; Remove["Global`*"]; DateList[Date[]] // Most
```
Out[54]=
```
{2020, 4, 27, 17, 8}
```

Discrete-Time Growth - Log utility function

We analyze the discrete-time growth model with Cobb-Douglas production function and quadratic utility.
We choose parameters so that the steady state is k=1.

In[55]:=
```
β = .95; α = .25; f[x_] = x + A x^α;
A=(1-β)/(α β); kss=1.; css = f[1]-1;
```

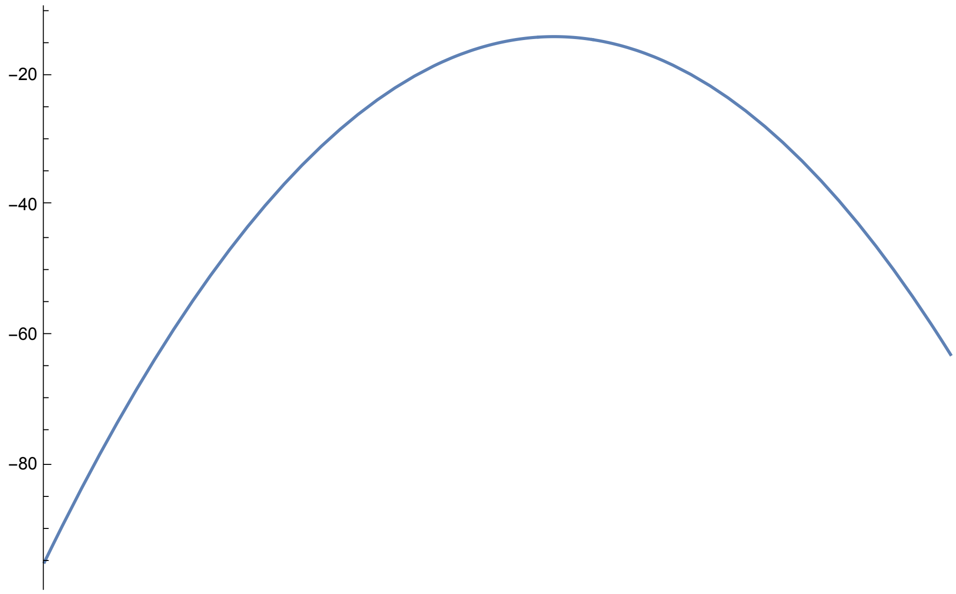
We choose the log utility function

```
util[x_] = Log[10., x];
negc = Series[util[x], {x, .01, 2}] // Normal;
u[x_] = If[x > .01, util[x], Evaluate[negc]]
```

Choose initial guess

In[61]:=
```
valinit[x_] = -14 - 100 (x - 1)^2;
Plot[valinit[x], {x, 0.1`, 1.7`}]
```

Out[62]=



Unstable example

In[63]:=
```
xmin = 0.1; xmax = 1.5;
```

Specify nodes

In[64]:=
```
nodes = Table[x, {x, .2, 1.5, .1}];
Length[nodes]
```

Out[65]=
```
14
```

Define newval[x] which computes the new value of of V[x] given by the RHS of the Bellman equation.

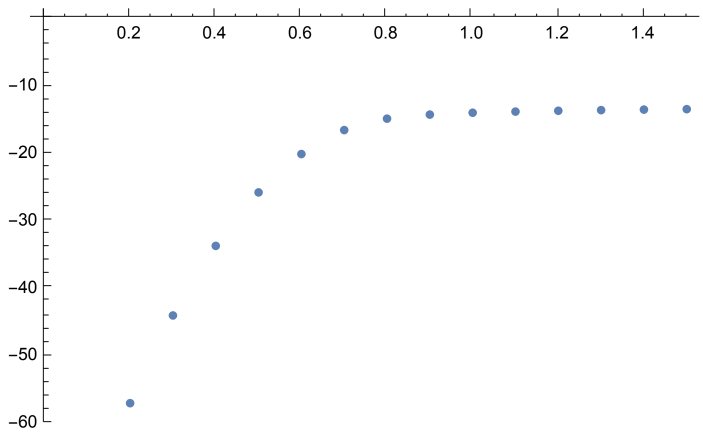
In[66]:=
```
newval[x_] := FindMaximum[
   u[c] + β val[f[x] - c], (* Objective *)
   {c, css}, (* Initial guess *)
   AccuracyGoal → 6][[1]]
```

Set val[x] to the initial guess and do first VFI

In[67]:=
```
val[x_] = valinit[x];
```

In[68]:=
```
newvs = Table[{nodes[[i]], newval[nodes[[i]]]}, {i, 1, Length[nodes]}];
ListPlot[newvs]
```

Out[69]=

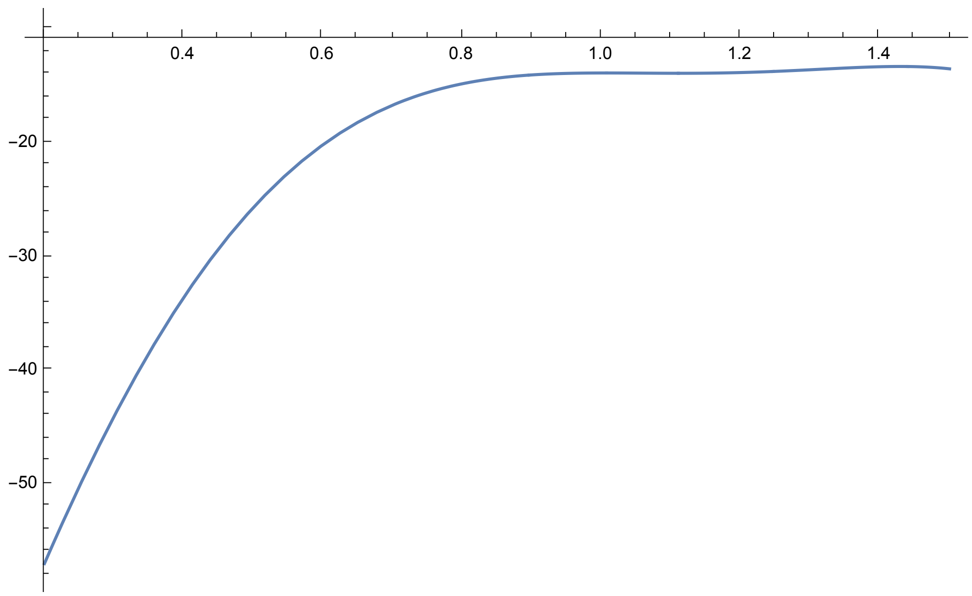


Compute new value function

In[70]:=
```
powers = Table[x^i, {i, 0, 5}];
Clear[val]; val[x_] = Fit[newvs, powers, x]
Plot[val[x], {x, 0.2`, 1.5`}, PlotRange → All]
```

Out[71]= $-88.6899 + 168.997\,x - 17.2121\,x^2 - 223.862\,x^3 + 197.348\,x^4 - 50.453\,x^5$

Out[72]=



Define a value function iteration command

In[73]:=
```
vfi :=
 (newvs = Table[{nodes[[i]], newval[nodes[[i]]]}, {i, 1, Length[nodes]}];
  Clear[val]; val[x_] = Fit[newvs, powers, x];
  Plot[val[x], {x, 0.2`, 1.5`}, PlotRange → All])
```

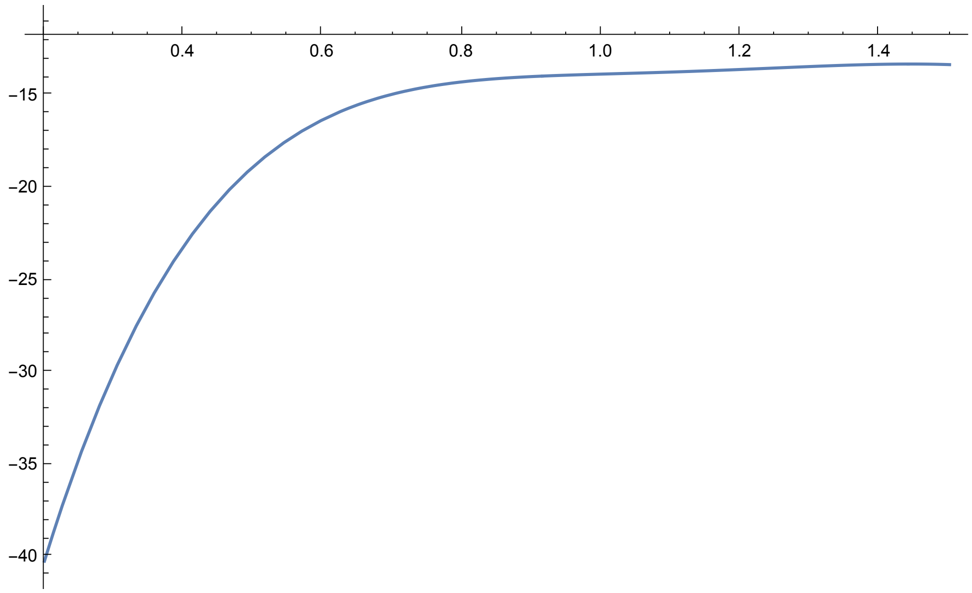
In[74]:=
```
iter = 1;
```

In[75]:=
```
Print["iteration number:"]; iter = iter + 1
vfi
```

```
iteration number:
```

Out[75]= 2

Out[76]=

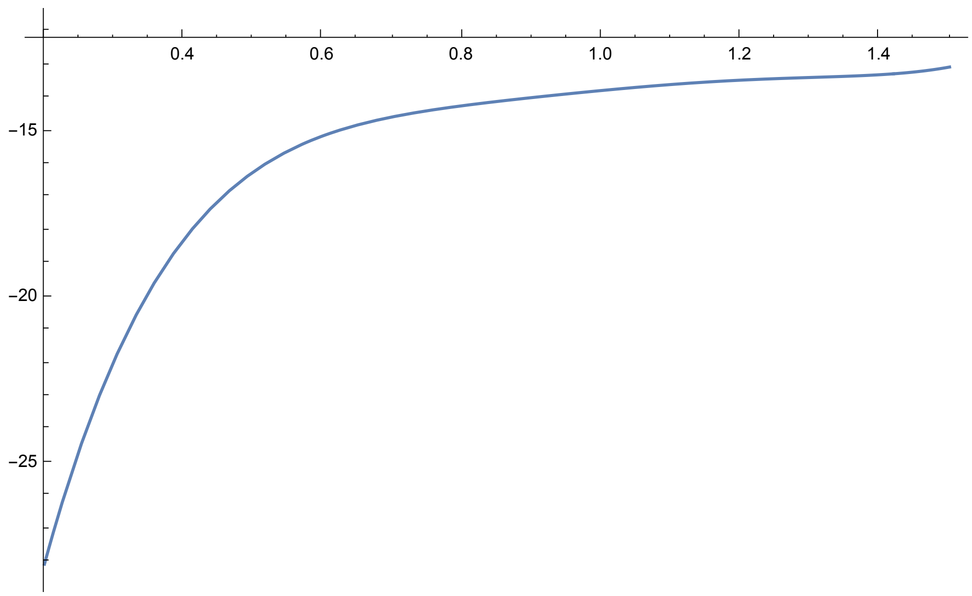


In[77]:=
```
Print["iteration number:"]; iter = iter + 1
vfi
```

```
iteration number:
```

Out[77]= 3

Out[78]=

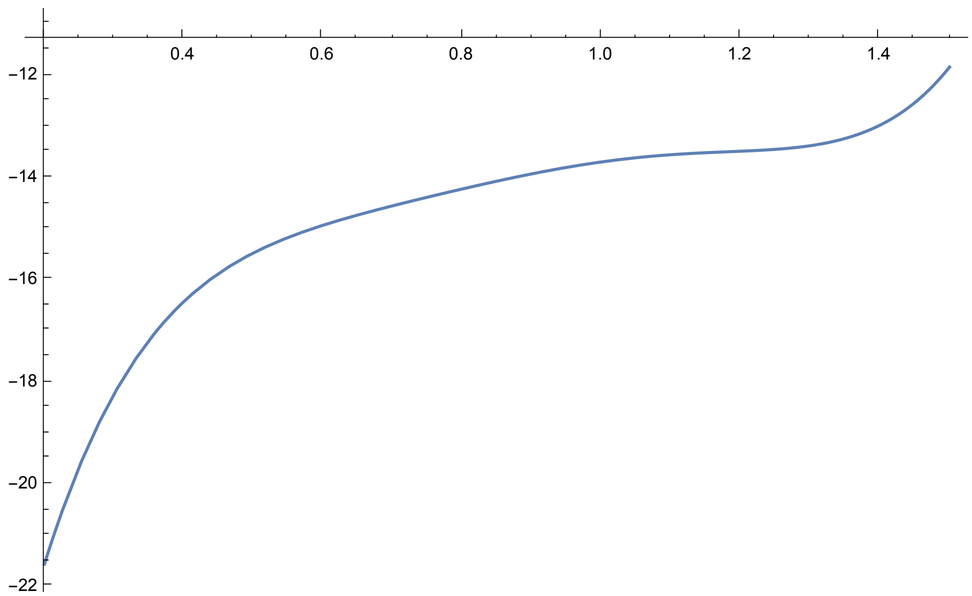


In[79]:=
```
Print["iteration number:"]; iter = iter + 1
vfi
```

```
iteration number:
```

Out[79]= 4

Out[80]=

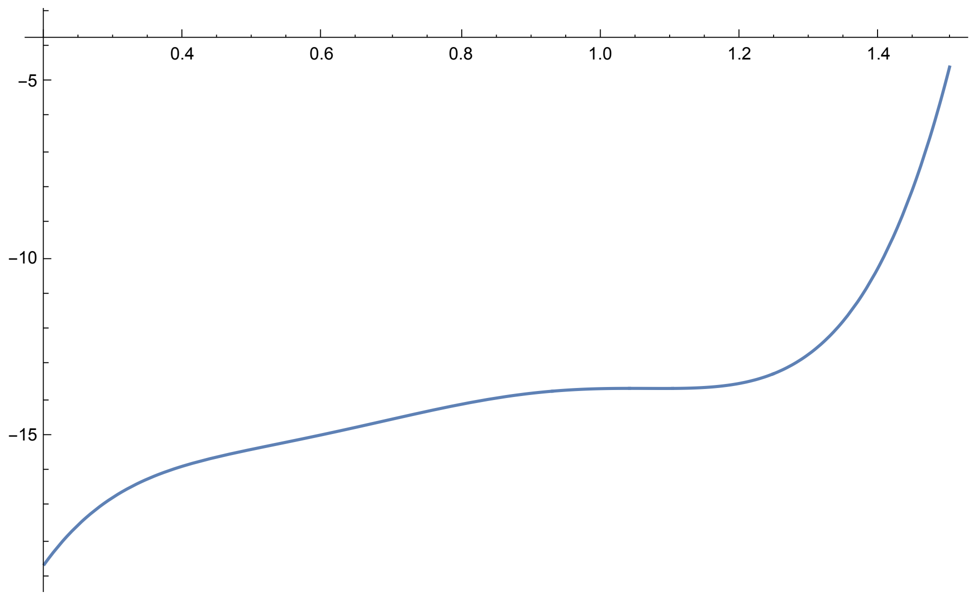


In[81]:=
```
Print["iteration number:"]; iter = iter + 1
vfi
```

```
iteration number:
```

Out[81]= 5

Out[82]=

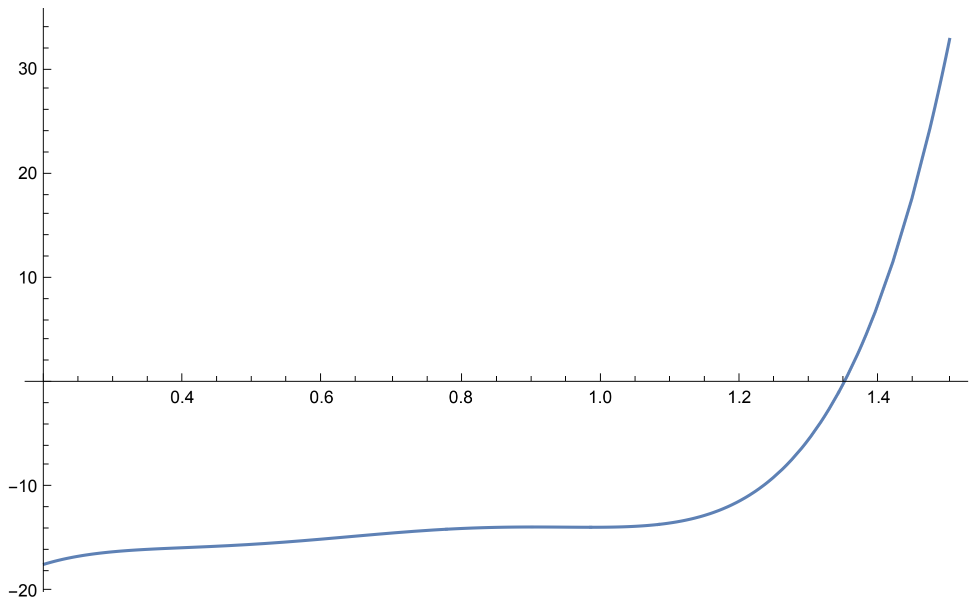


In[83]:=
```
Print["iteration number:"]; iter = iter + 1
vfi
```

```
iteration number:
```

Out[83]= 6

Out[84]=



There is no next value function. The reason is that the previous iterate was sufficiently convex that some Bellman optimization problems were unbounded.

Stabilize with constraint

Specify nodes

In[85]:=
```
nodes = Table[x, {x, .2, 1.5, .1}];
Length[nodes]
```

Out[86]= 14

Define newval[x] which computes the new value of of V[x] given by the RHS of the Bellman equation.

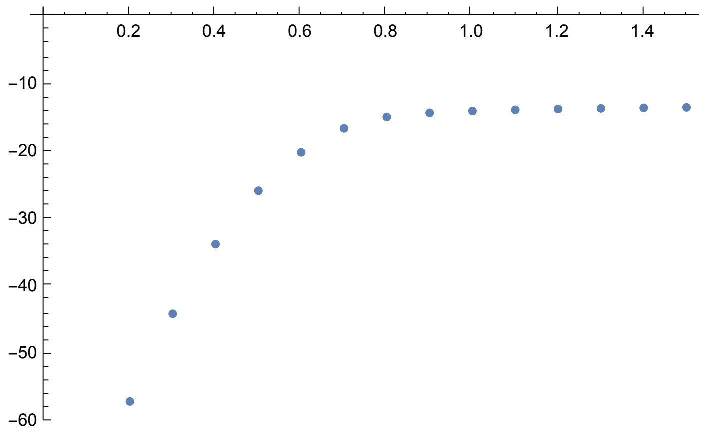
In[87]:=
```
newval[x_] := FindMaximum[
   {u[c] + β val[f[x] - c], xmin ≤ f[x] - c ≤ xmax} , (* Objective and constraint *)
   {c, css}, (* Initial guess *)
   AccuracyGoal → 6][[1]]
```

Set val[x] to the initial guess and do first VFI

In[88]:=
```
val[x_] = valinit[x];
```

In[89]:=
```
newvs = Table[{nodes[[i]], newval[nodes[[i]]]}, {i, 1, Length[nodes]}];
ListPlot[newvs]
```

Out[90]=

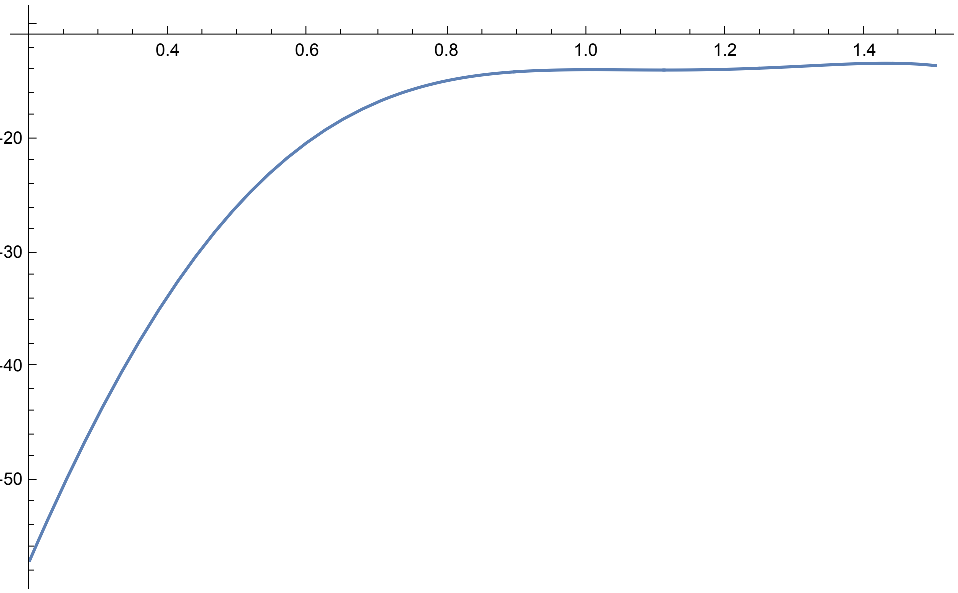


Compute new value function

In[91]:=
```
powers = Table[x^i, {i, 0, 5}];
Clear[val]; val[x_] = Fit[newvs, powers, x]
Plot[val[x], {x, 0.2`, 1.5`}, PlotRange → All]
```

Out[92]= $-88.6899 + 168.997\,x - 17.2121\,x^2 - 223.862\,x^3 + 197.348\,x^4 - 50.453\,x^5$

Out[93]=



Define a value function iteration command

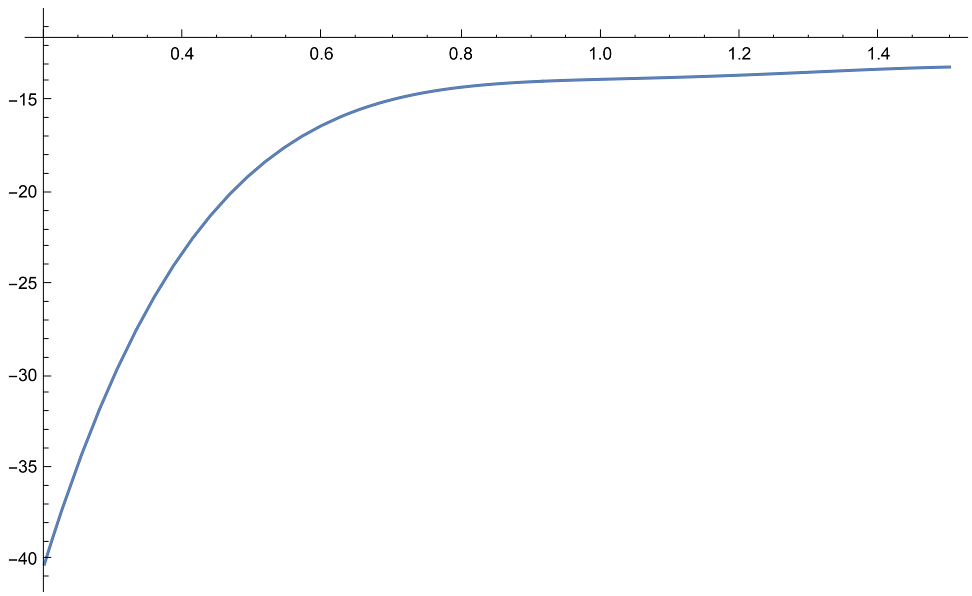
```
In[94]:= vfi :=
  (newvs = Table[{nodes[[i]], newval[nodes[[i]]]}, {i, 1, Length[nodes]}];
   Clear[val]; val[x_] = Fit[newvs, powers, x];
   Plot[val[x], {x, 0.2`, 1.5`}, PlotRange → All])

In[95]:= iter = 1;
```

In[96]:=
```
Print["iteration number:"]; iter = iter + 1
vfi
```

```
iteration number:
```

Out[96]= 2

Out[97]=

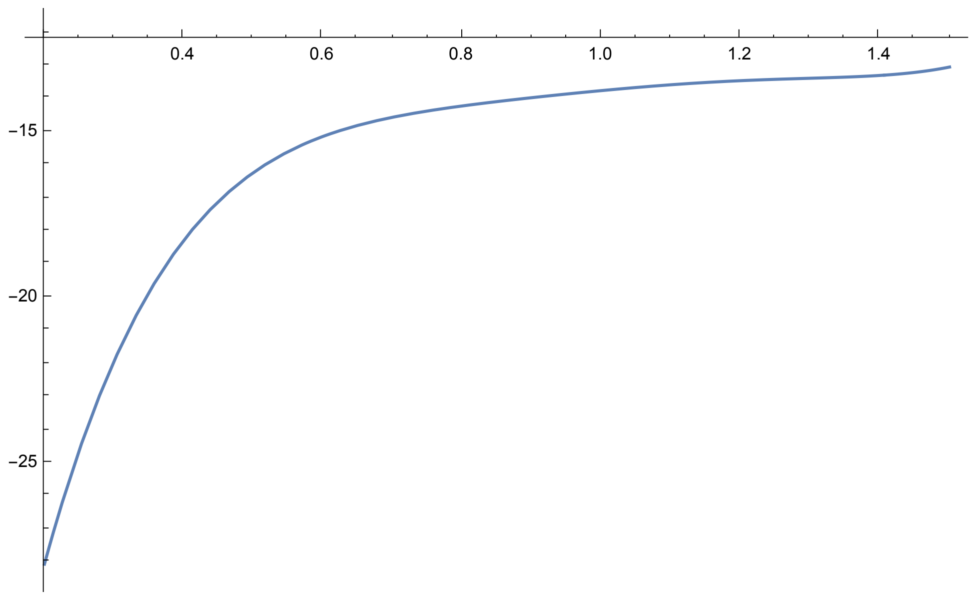


In[98]:=
```
Print["iteration number:"]; iter = iter + 1
vfi
```

```
iteration number:
```

Out[98]= 3

Out[99]=

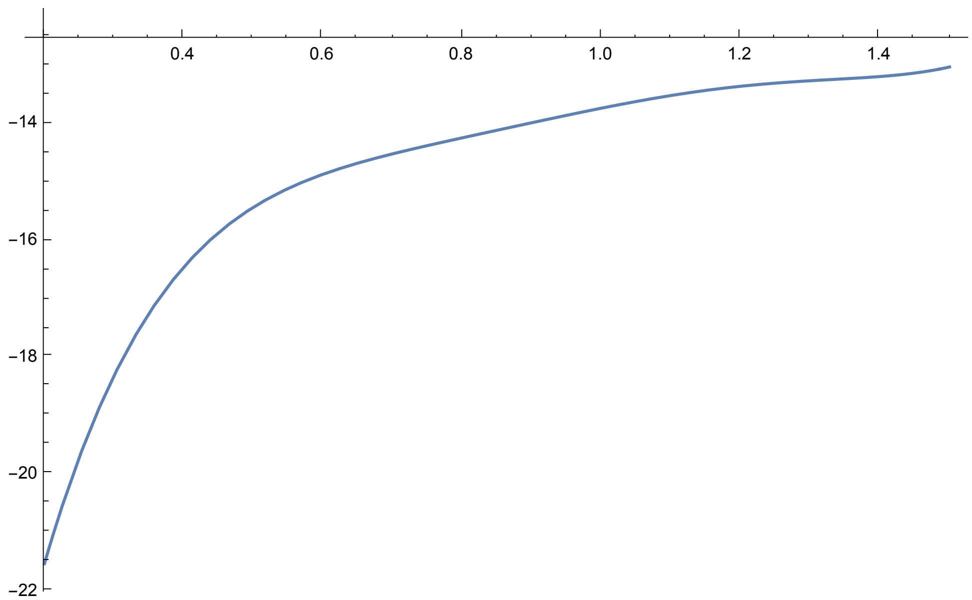


In[100]:=
```
Print["iteration number:"]; iter = iter + 1
vfi
```

```
iteration number:
```

Out[100]= 4

Out[101]=

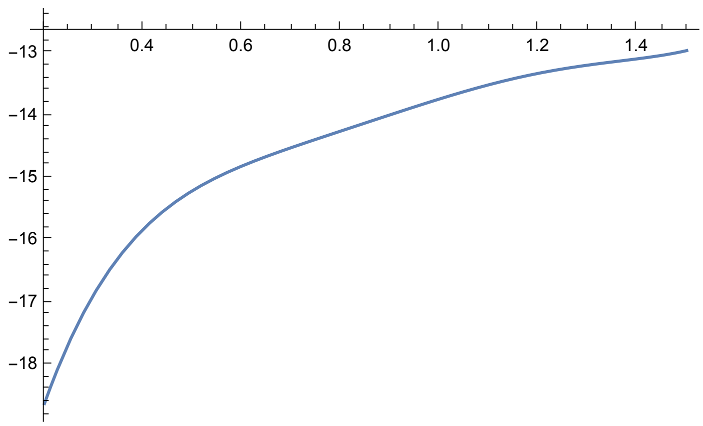


In[102]:=
```
Print["iteration number:"]; iter = iter + 1
vfi
```

iteration number:

Out[102]= 5

Out[103]=

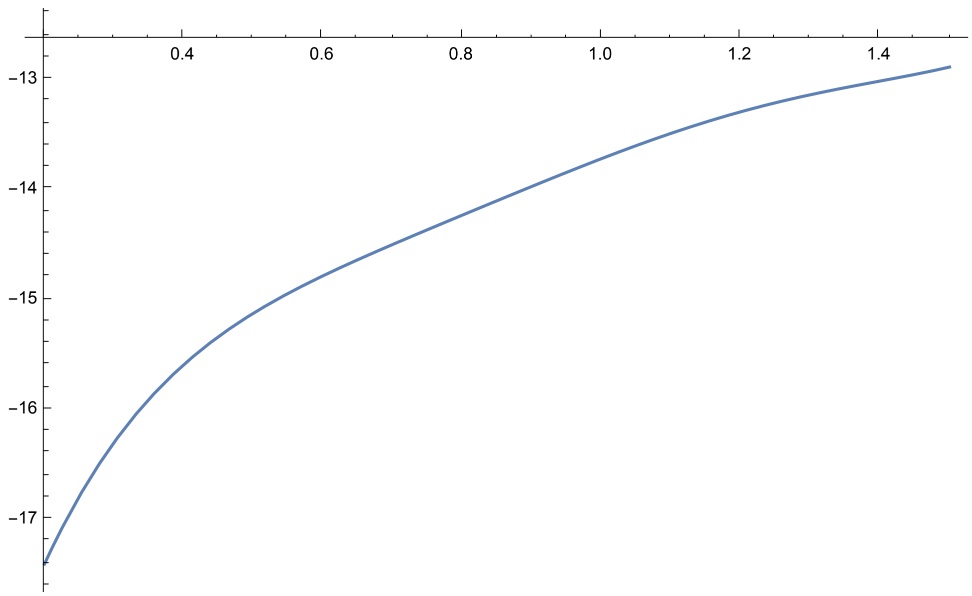


In[104]:=
```
Print["iteration number:"]; iter = iter + 1
vfi
```

iteration number:

Out[104]= 6

Out[105]=

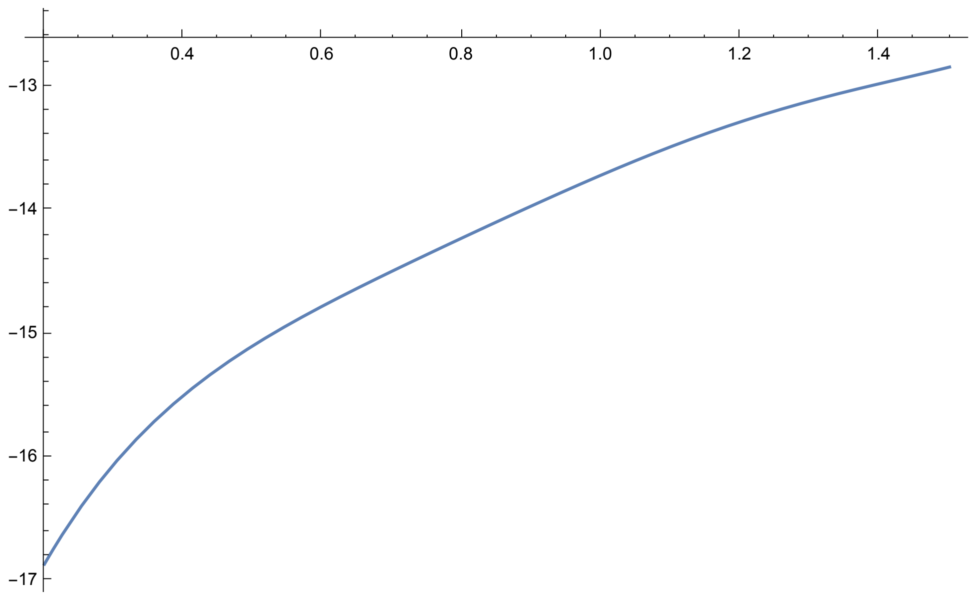


In[106]:=
```
Print["iteration number:"]; iter = iter + 1
vfi
```

```
iteration number:
```

Out[106]= 7

Out[107]=

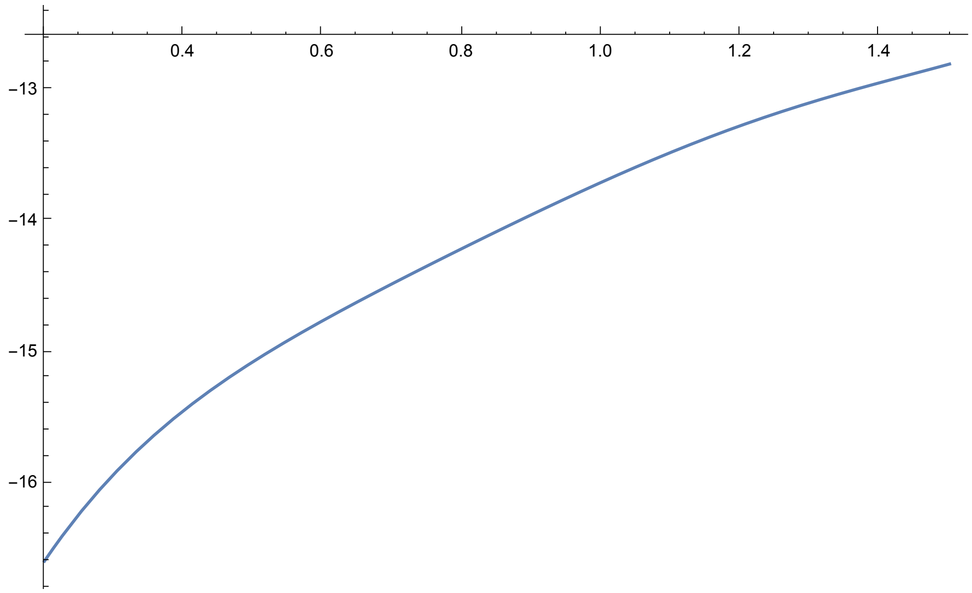


In[108]:=
```
Print["iteration number:"]; iter = iter + 1
vfi
```

```
iteration number:
```

Out[108]= 8

Out[109]=

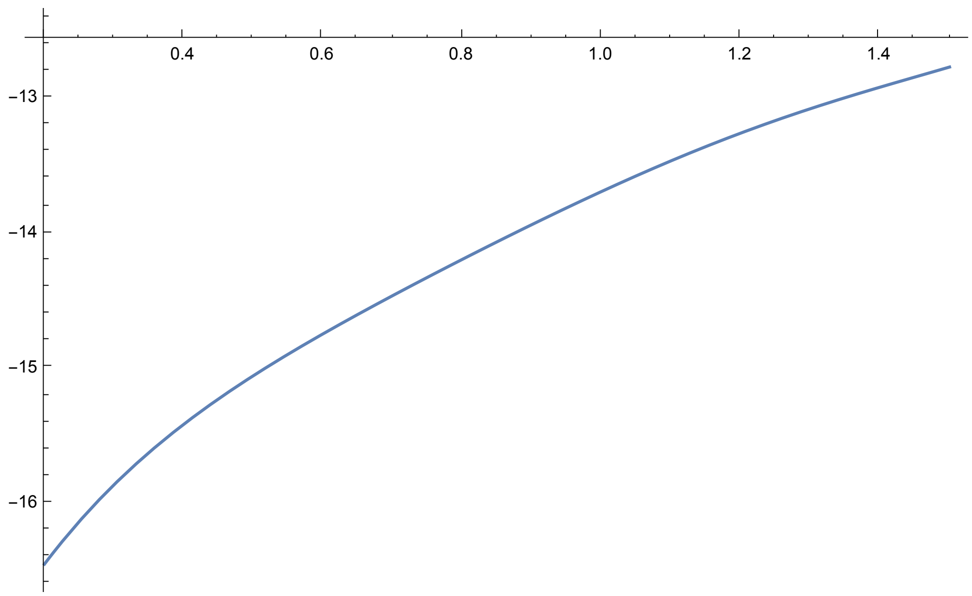


In[110]:=
```
Print["iteration number:"]; iter = iter + 1
vfi
```

iteration number:

Out[110]= 9

Out[111]=

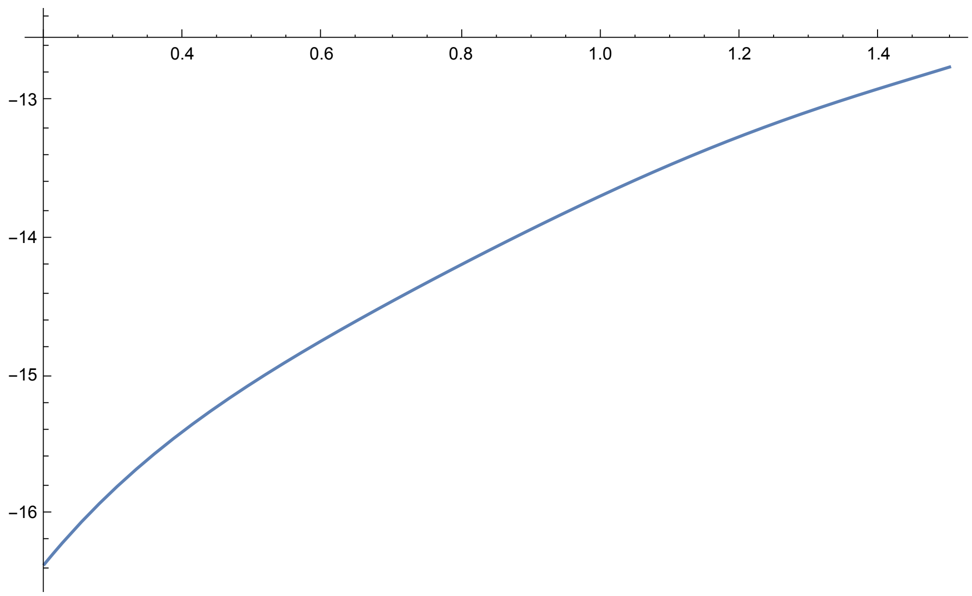


In[112]:=
```
Print["iteration number:"]; iter = iter + 1
vfi
```

iteration number:

Out[112]= 10

Out[113]=

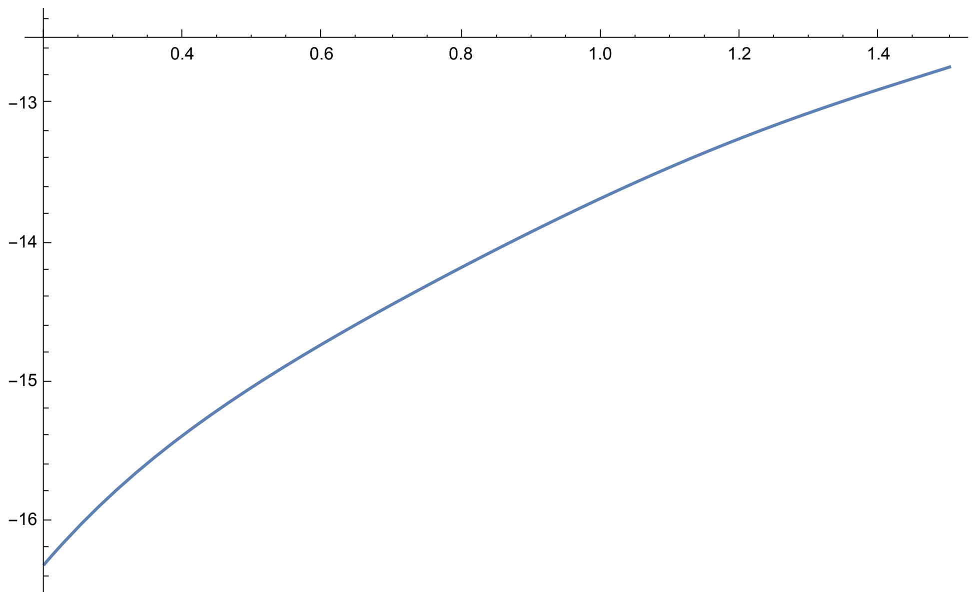


In[114]:=
```
Print["iteration number:"]; iter = iter + 1
vfi
```

```
iteration number:
```

Out[114]= 11

Out[115]=

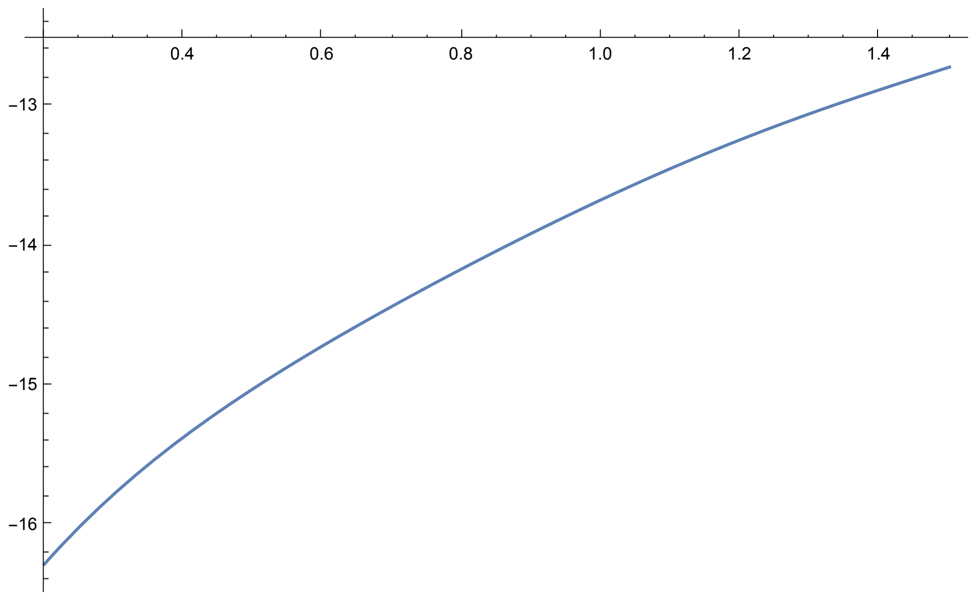


In[116]:=
```
Print["iteration number:"]; iter = iter + 1
vfi
```

```
iteration number:
```

Out[116]= 12

Out[117]=

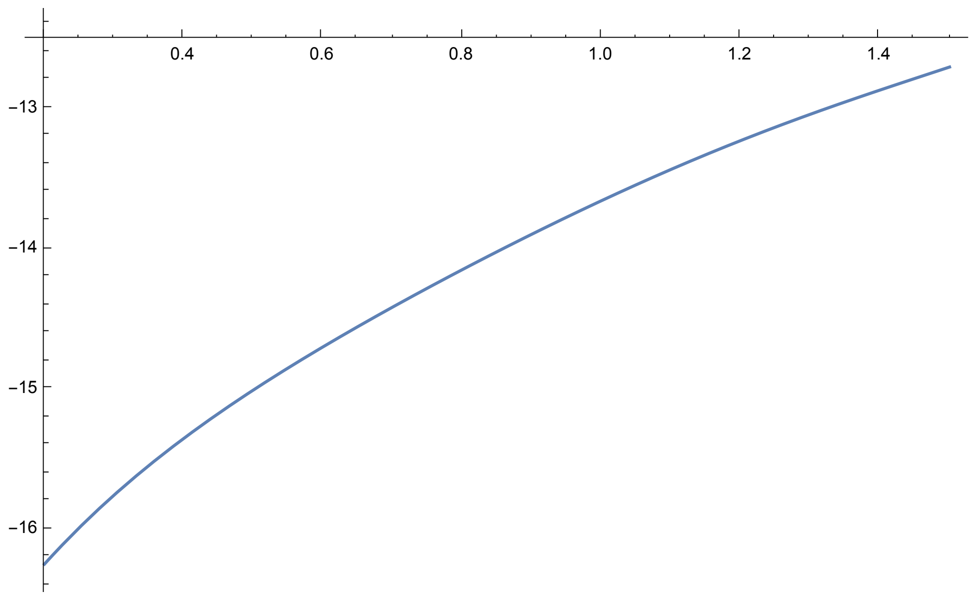


In[118]:=
```
Print["iteration number:"]; iter = iter + 1
vfi
```

```
iteration number:
```

Out[118]= 13

Out[119]=

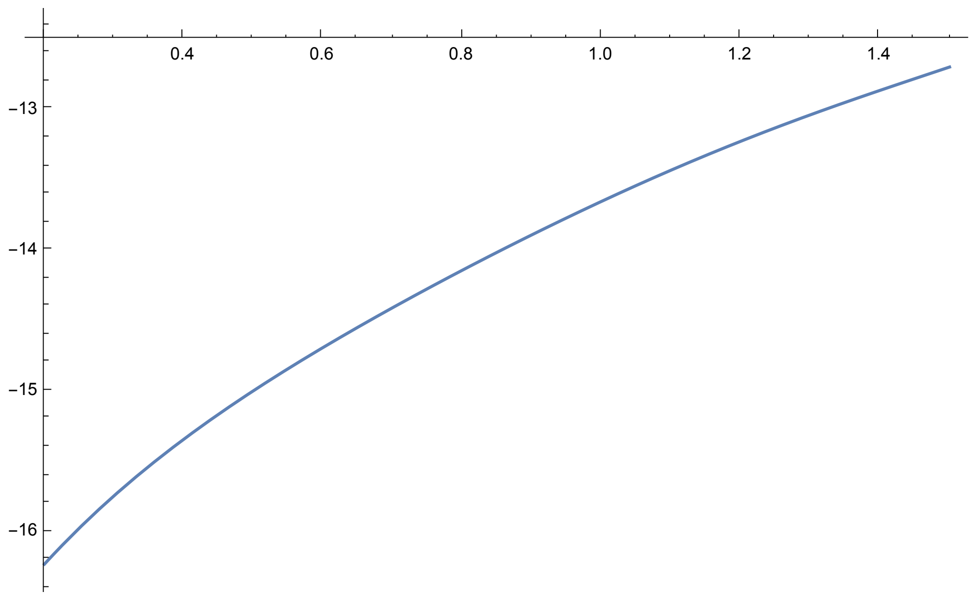


In[120]:=
```
Print["iteration number:"]; iter = iter + 1
vfi
```

```
iteration number:
```

Out[120]= 14

Out[121]=

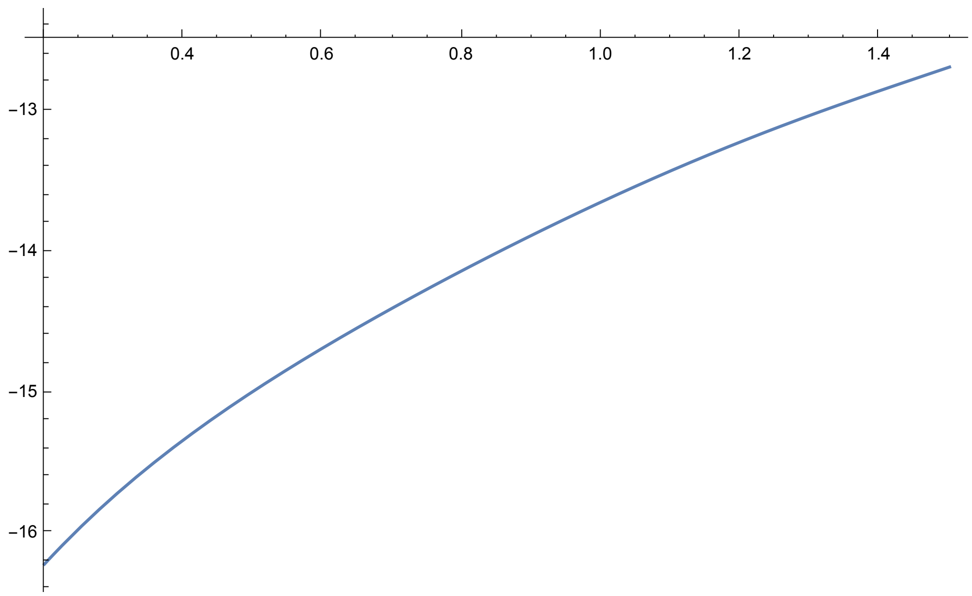


In[122]:=
```
Print["iteration number:"]; iter = iter + 1
vfi
```

```
iteration number:
```

Out[122]= 15

Out[123]=



Lesson

Thou shalt not extrapolate.

- Value function has no meaning outside the domain of definition
- Extrapolations are often crazy

Add constraints forcing the state in the next period to be in the set of permissible states for the next period.