Optimization: Three-period Life-Cycle problem

In[*]:=
```
x = 0; Remove["Global`*"]; Date[]
```

Out[*]=
```
{2019, 1, 15, 13, 34, 7.376712}
```

This was run under Mathematica version 7. This notebook now runs correctly, but the errors are still instructive.

Basic three period problem

Here is a three-period objective with discount factor β

```
In[•]:= obj = u[c1] + β u[c2] + β^2 u[c3]
Out[•]= u[c1] + β u[c2] + β^2 u[c3]
```

Let R be the gross return on savings, The present values of consumption expenditures and wage income are

```
In[•]:= PVc = c1 + c2 / R  + c3 / R^2 ;
        PVw = w1 + w2 / R  + w3 / R^2;
```

Specify the income path. We choose a pattern that corresponds to a life-cycle,

```
In[•]:= w1 = 2; w2 = 3; w3 = 0;
```

Choose an interest rate, r, gross return, R, and a discount factor, β

```
In[•]:= r = 0.2; R = 1 + r;  β = 0.9;
```

We first assume an exponential utility function

```
In[•]:= u[c_] = -Exp[-c];
```

The budget constraint says that the present value of consumption equals the present value of wage income.

```
In[•]:= budget = {PVc == PVw};
```

We want to solve the life-cycle maximization problem for this individual.
List the variables

```
In[•]:= vars = {c1, c2, c3};
```

Display the resulting life-cycle utility

```
In[•]:= obj
Out[•]= -ℯ^-c1 - 0.9 ℯ^-c2 - 0.81 ℯ^-c3
```

Display the constraint

In[•]:=
```
budget
```
Out[•]=
```
{c1 + 0.833333 c2 + 0.694444 c3 == 4.5}
```

We have specified the objective (obj), the set of constraints (budget), and the variables (vars). We now call the FindMaximum command.

In[•]:=
```
FindMaximum[{obj, budget}, vars]
```
Out[•]=
```
{-0.456019, {c1 → 1.71256, c2 → 1.78952, c3 → 1.86648}}
```

Domain Problem

In[•]:=
```
x = 0; Remove["Global`*"]
```

Let's try an example which illustrates the problem that the solver may evaluate the objective at points where the objective is not defined. [NOTE: This is being run under *Mathematica* 7. The specifications are Log[c], w=(1, 1, 0), R=1.2, and β=1. I note this here in case future versions of Mathematica work.]

We specify Log utility, and define other details

```
u[c_] = Log[c];
obj = u[c1] + β u[c2] + β^2 u[c3];

PVc := c1 + c2 / R + c3 / R^2;
PVw := w1 + w2 / R + w3 / R^2;

budget := {PVc == PVw};

w1 = 0; w2 = 0; w3 = 1;
R = 12 / 10;
β = 1;
```

Out[•]=
$$\text{Log}[c1] + \beta\, \text{Log}[c2] + \beta^2\, \text{Log}[c3]$$

Display the elements of the optimization problem

In[•]:=
```
vars = {c1, c2, c3};
```

In[•]:=
```
obj
```

Out[•]=
$$\text{Log}[c1] + \text{Log}[c2] + \text{Log}[c3]$$

In[•]:=
```
budget
```

Out[•]=
$$\left\{c1 + \frac{5\, c2}{6} + \frac{25\, c3}{36} == \frac{25}{36}\right\}$$

Solve the life-cycle problem with this utility function.

In[•]:=
```
FindMaximum[{obj, budget}, vars]
```

FindMaximum: The function value 2.48188 – 3.14159 *i* is not a real number at {c1, c2, c3} = {–0.24548, 0.378766, 0.898972}.

IPOPTMinimize: Invalid objective function. The objective function doesn't evaluate to a real–valued numeric result at the initial point.

FindMaximum: The function value 2.48188 – 3.14159 *i* is not a real number at {c1, c2, c3} = {–0.24548, 0.378766, 0.898972}.

FindMaximum: The function value 2.48188 – 3.14159 *i* is not a real number at {c1, c2, c3} = {–0.24548, 0.378766, 0.898972}.

General: Further output of FindMaximum::nrnum will be suppressed during this calculation.

FindMaximum: The algorithm does not converge to the tolerance of 1.`*^–6 in 500 iterations. The best estimated solution, with feasibility residual, KKT residual, or complementary residual of $\{3.31402 \times 10^{-16}, 4.9106, 0\}$, is returned.

Out[•]=
```
FindMaximum[{obj, budget}, vars]
```

This failed because the algorithm evaluated the utility function at a negative consumption. We need to avoid this. To do so, we add constraints forcing the variables to remain positive.

Good initial guess

One way to avoid the domain problem is to give the algorithm an initial guess good enough for the convergence theorems to apply. We next define an initial guess

In[•]:=
```
init = {{c1, 1 / 4}, {c2, 1 / 4}, {c3, 1 / 4}};
FindMaximum[{obj, budget}, init]
```

Out[•]=
```
{-3.8428, {c1 → 0.231481, c2 → 0.277778, c3 → 0.333333}}
```

That was a feasible initial guess. Next, let's try an infeasible initial guess

In[•]:=
```
init = {{c1, 1}, {c2, 1}, {c3, 1}};
FindMaximum[{obj, budget}, init]
```

Out[•]=
```
$Aborted
```

That did not go well. I had to kill it.

Bounds on variables

We often don't have good initial guesses. We need a more robust strategy.
Let's put constraints on the consumption vector, hoping that no iterate will violate them. This is not guaranteed in general since the only requirement is that the constraints be satisfied at the solution, not along the path to the solution.

In[•]:=
```
lbnds = {c1 ≥ 0.0001, c2 ≥ 0.0001, c3 ≥ 0.0001};
```

Now we form a new constraint set combining the budget constraint with the positivity constraints.

In[•]:=
```
constraints = Union[budget, lbnds]
```

Out[•]=
$$\left\{c1 + \frac{5\ c2}{6} + \frac{25\ c3}{36} == \frac{25}{36},\ c1 \geq 0.0001,\ c2 \geq 0.0001,\ c3 \geq 0.0001\right\}$$

We now try to solve the problem with the additional constraints

In[•]:=
```
FindMaximum[{obj, constraints}, vars]
```

Out[•]=
```
{-3.8428, {c1 → 0.231481, c2 → 0.277778, c3 → 0.333333}}
```

Adding bounds helped in this case, and generally improves reliability.

Combine bounds and good initial guess

Of course, the best thing to do is to both impose lower bounds on the variables and give it a good initial guess

In[•]:=
```
init = {{c1, 1 / 4}, {c2, 1 / 4}, {c3, 1 / 4}};
FindMaximum[{obj, constraints}, init]
```

Out[•]=
```
{-3.8428, {c1 → 0.231481, c2 → 0.277778, c3 → 0.333333}}
```