Computational Economics 2020

Assignment 2

(due: Wednesday, May 20, 2020, 11:59 PM (GMT + 1))

The assignment is split into part A and part B which are weighted equally with 15% each. You are allowed to form groups of 2 (if there is a odd number of people, groups of 3 are allowed too). Each group has to present at least one of their solutions.

Please follow these instructions for handing in your assignment:

1. Submit the assignment via Email to Philipp (philipp.mueller@business.uzh.ch). The Email should contain:
   - A single PDF-file with the names of all group members, and assignment number on page 1. The file should contain all your answers and results.
   - The source code in a separate zip archive. The code should be well documented and readable.
2. Only the students taking the course for credits are getting feedback to their solution; feel free to send your solution anyway. A sample solution will be published after the deadline. If you have any specific question, reach out on GitHub `https://github.com/KennethJudd/CompEcon2020/issues`.

Refrain from sharing complete solutions.
For the content of the PDF we expect the following:

1. Provide a brief introduction/ motivation for the problem.
2. Explain how you solved the exercise and show the most relevant calculations (formulas and essential parts of the code) with brief comments.
3. Concisely interpret the results of the exercise.
4. For each exercise the floating text should not exceed 2 pages (this does not include formulas, codes, graphs, and tables).

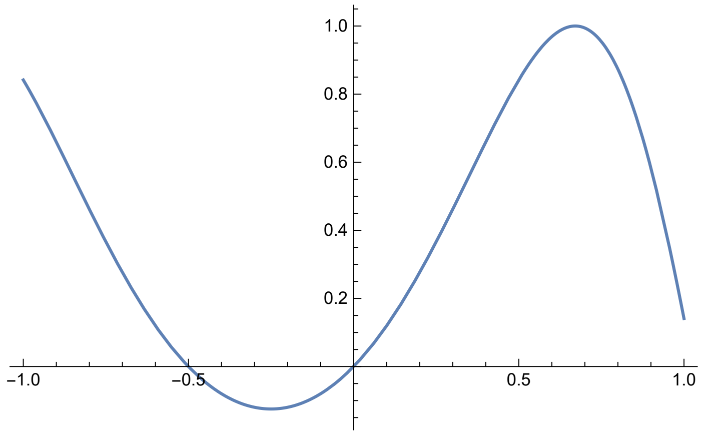
Exercise A-1: Approximation Problem

Consider the following function

```
f[x_] = Sin[x + 2 x^2]
Plot[f[x], {x, -1, 1}]
```

Out[ ]= $\mathrm{Sin}[x + 2x^2]$

Out[ ]=



Compute Chebyshev polynomial interpolation of f[x] using 5, 10, and 15 Chebyshev points. (This means, express the interpolation in terms of Chebyshev polynomials)

Compute polynomial interpolation of f[x] using 5, 10, and 15 uniformly distributed points. (This means, express the interpolation in terms of ordinary polynomials)

Compute degree 5, 10, and 15 least squares polynomial approximation of f[x] using 10, 20, and 30 uniformly distributed points.
(This means use 10 points for the degree 5 approximation, 20 points for the degree 10 approximation, etc. I am asking for three approximations not nine.)

Compute degree 5, 10, and 15 least squares polynomial approximation of f[x] using 10, 20, and 30 Chebyshev zeros.

Compute degree 5, 10, and 15 least squares polynomial approximation of f[x] using 10, 20, and 30 random points uniformly distributed on the interval [-1,1].
I need to be more specific. First draw 10 random points, and use that for the degree 5 approximation. Then draw an independent 20 points, and use that set for the degree 5 approximation. ....

For each approximation, compute the L^2, L^1, and L^∞ norms of the error.

Compare the errors of the degree d approximations, for d=5, 10, 15.

So, for f[x], you should
- plot the function
- construct a table that displays the norms of the errors

I will help by being specific about the table. Let the different rows correspond to different approximation methods, and the different columns correspond to the different norms.

Write a program which takes f[x] as input and then produces the plot and the table as output.

Run the program for each of the following functions:

$\mathrm{Sin}\left[x + 2x^2\right]$
$\mathrm{Log}[x + 1.1]$
$\mathrm{Log}\left[10\,(x + 1.01)\right]$
$(x + 1.1)^{1/4}$
$(x + 1.1)^{1/10}$
$(x + 1.01)^{1/4}$
$(x + 1.01)^{1/10}$
$(x + 1.01)^{-2}$
$(x + 1.01)^{-4}$
$(1 - 3x + e x^2)^{-2/3}$

Exercise A2: Gauss-Laguerre quadrature

Compute the discounted utility integral $\int_0^\infty e^{-\rho t}\, u(c(t), l(t))\, dt$ where $c(t) = 1 - e^{\mu_1 t}$, $l(t) = 1 - e^{-\mu_2 t}$, and $u(c, l) = (c^\sigma + l^\sigma)^{\frac{\gamma+1}{\sigma}} \Big/ (\gamma + 1)$. Let $\gamma \in \{-0.5, -1.1, -3, -10\}$, $\sigma \in \{0.5, 2.0\}$, $\mu_1 \in \{0.05, 0.1, 0.2\}$, $\mu_2 \in \{0.05, 0.1, 0.2\}$, ρ=0.05.
Use Gauss-Laguerre rules. How do the choices of γ, σ, μ_1, and μ_2 affect accuracy? Use 20 Gauss-Laguerre quadrature nodes. For the worst performing configuration, evaluate and plot the convergence w.r.t. the number of quadrature points (e.g., for {5,10,15, 20, 25, ...}) . Compare the accuracy and running time to Monte-Carlo integration with n ∈ {10,100, ..., 10 ^k} nodes.

Exercise A-3

Consider the life-cycle optimization problem from the last assignment

$$\max_{\{c_t\}_{t=1}^{T},\{a_t\}_{t=1}^{T+1}} \sum_{t=1}^{T} \beta^t \log(c_t) \tag{1}$$

$$\text{s.t. } a_{t+1} \leq (1+r)a_t + w_t - c_t \tag{2}$$

$$a_{T+1} \geq 0 \tag{3}$$

$$c_t > 0 \tag{4}$$

and earns

$$w_t = \begin{cases} \max\left(1.5, (0.5 + t(1-t/T))\right)/16 & t < T_{retirement} \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

(A) Solve the constrained optimization problem with $\beta = 0.96$, $r = 0.1$, $T = 50$, $T_{retirement} = 0.9T$ and $a_1 = 1$.

(B) Use the automatic differentiation tool of your choice to solve the life cycle optimization problem. We recommend using CasADi which is available for Python as well as Matlab. To help to get you started here a sample code:

```python
from casadi import *
# Define the variables you are optimizing for as
# CasADi symbolic variables
x=MX.sym('x', 2, 1)
y=MX.sym('y', 1, 1)
# Define the objective function note that it is a symbolic expression
f=x[0]**2+ x[1]+100*y
# Define the constraint(s)
g=(1-x)**2-y

# create a dictionary with the following keys:
# x = vector of optimization variables
# f = objective function
# g = (non)-linear constraints (can be a vector as well)
P = dict(x=vertcat(x,y),f=f,g=g)
# Create solver instance and pass the dict as argument
F=nlpsol('F','ipopt',P) #'ipopt' is an interior point optimizer

# Solve the problem
# ubg and lbg denote the upper and lower bound for the constraints
# ubx and lbx would denote the upper and lower bounds for the variables
r=F(x0=[2.5,3.0,0.75],ubg=0,lbg=0)
print(r['x'].full()) #solution
```

Listing 1: CasADi Python example

(C) Run the optimization for $T \in \{10, 20, 40, 80, 160, 320, 640, 1280, \dots\}$ for FD and automatic differentiation compare their wall times, number of iterations, and first-order optimality. If you have solved last time for the analytic derivatives, use them as additional comparison. As discussed in the lecture, keep β^T and r^T constant by adjusting β and r.