5 Optimering i Julia

Vi kommer til at løse de opgaver, der svarer til problemerne *Incredible Chairs 1, 2, 3, Class Jobs 1, 2* og *Groceries 1,2* i **Mathematical Programming with Julia**, som er at finde på

`https://www.man.dtu.dk/mathprogrammingwithjulia#home`

5.1 Incredible Chairs

Firmaet *Incredible Chairs* producerer to typer af stole, A og B, hvor én Stol A kan sælges med en profit på 4 og én Stol B kan sælges med en profit på 6. Stolene produceres på 3 samlebånd:

1. Bånd 1 kræver 2 timer for at producere en enhed af Stol A, og der kan ikke bruges mere end 14 timer grundet anden produktion på båndet.
2. Bånd 2 kræver 3 timer for at producere en enhed af Stol B, og der kan ikke bruges mere end 15 timer.
3. Bånd 3 kræver 4 timer for at producere en enhed af Stol A, 3 timer for at producere en enhed af Stol B, og der kan ikke bruges mere end 36 timer.

Firmaet ønsker at maksimere den samlede profit, givet de ovenfor nævnte bibetingelser:

a. Opstil firmaets problem matematisk.

I filen `IncredibleChairs.jl` finder du en implementering af problemet.

b. Kør koden og tjek dit resultat.

Nedenfor ser I nogle nyttige kommandoer i Julia, som kan bruges efter en optimering:

```
# Værdi af variabel i løsning
value(xA)

# Hent tal til sensitivitetsanalyse:
report = lp_sensitivity_report(IC)

# Tolerance ned og op for koefficient i objektfunktionen
report[xA]

# Tolerance ned og op for grænse i bibetingelsen
report[c1]

# Skyggepris for bibetingelse
shadow_price(c1)

# Alternativomkostning for koefficient i objektfunktionen
reduced_cost(xA)
```

```
# Grænse for bibetingelse (højreside i bibetingelse)
normalized_rhs(c1)

# Værdi af venstreside i bibetingelse
value(c1)

# Værdi af koefficient i objektfunktionen
coefficient(objective_function(IC), xA)
```

c. Beregn værdien af slack-variablene ved brug af kommandoerne ovenfor i Julia, og foretag en sensitivitetsanalyse på grænserne i bibetingelse og tilsvarende på koefficienterne i objektfunktionen.

d. Omformulér problemet så kun heltallige værdier af stole bliver overvejet (dvs. til et 'mixed integer' problem). Ændr i koden ovenfor og find løsningen til det nye problem.

I filen `IncredibleChairs2.jl` finder du en mere generel tiltang til problemet.

e. Gentag b. og d. med den alternative kode. Er der nogen forskel til den forrige model? Hvilken tilgang foretrækker I?

Incredible Chairs har besluttet at udvide deres porteføljе til 10 stole, og har i samme omfang udvidet til 5 samlebånd. Nedenfor ses data for profitte for hver stol solgt

	Stoltype									
	A	B	C	D	E	F	G	H	I	J
Profit (E)	6	5	9	5	6	3	4	7	4	3

og kapacitetsbegrænsningerne på samlebåndende

	Samlebånd				
	1	2	3	4	5
Samlebåndskapacitet	47	19	36	13	46

samt resurserne påkrævet for hver type stol på hver produktionsbånd

		Stoltype									
		A	B	C	D	E	F	G	H	I	J
Samlebånd	1	6	4	2	3	1	10	2	9	3	5
	2	5	6	1	1	7	2	9	1	8	6
	3	8	10	7	2	9	6	9	6	5	6
	4	8	4	8	10	5	4	1	5	3	5
	5	1	4	7	2	4	1	2	3	10	1

f. Tilpas koden fra d., og løs det nye profitmaksimeringsproblem for virksomheden.

5.2 Class Jobs

Din lillesøster og fire af hendes venner fra skolen har meldt sig til frivilligt arbejde. De fem børn $c \in C$ har fem jobs $j \in J$ at vælge i mellem. For at gøre det frivillige arbejde rarest muligt har man bedt hvert barn om at rangere jobbene fra 1 til 5 (med 5 det bedste). Denne information kan ses nedenfor, og angiver $\text{Ønske}_{j,c} \in \{1, 2, 3, 4, 5\}$.

		Barn				
		c_1	c_2	c_3	c_4	c_5
Job	j_1	1	3	2	5	5
	j_2	5	2	1	1	2
	j_3	1	5	1	1	1
	j_4	4	5	4	4	4
	j_5	3	5	3	5	3

eller mere kopieringsvenligt

```
    Ønske=[
1 3 2 5 5;
5 2 1 1 2;
1 5 1 1 1;
4 5 4 4 4;
3 5 3 5 3]
```

De fem børn tildeles nu hvert et job, og man vil gerne maksimere summen af rangeringerne af deres tildelte jobs.

a. Hvilken type problem er der tale om? Hvilke antagelser medfører det i problemet?

b. Løs problemet i Julia. Overholder løsningen alle begrænsninger?

Hint: Hvis I kalder jeres model `CJ` og bruger `x[j,c]` for $c \in C$ og $j \in J$ som variable, da kan man implementere bibetingelsen at hvert barn skal udføre netop ét job med kodestumpen

```
@constraint(CJ, one_job_pr_child[c=1:C],
    sum( x[j,c] for j=1:J) == 1
)
```

og tilsvarende at hvert job skal udføres af netop ét barn

```
@constraint(CJ, one_child_pr_job[j=1:J],
    sum( x[j,c] for c=1:C) == 1
)
```

Derudover kan man benytte sig af følgende kodestump til at printe svaret:

```
if termination_status(CJ) == MOI.OPTIMAL
    println("Optimal værdi af objektfunktionen: $(objective_value(CJ))")
    for c=1:C
        for j=1:J
```

```
            if value(x[j,c])>0.999
                println("Barn: ", c, " skal udføre job: ", j)
            end
        end
    end
else
    println("No optimal solution available")
end
```

Husk at ændre de relevante steder, hvis jeres modelnavn er anderledes eller I har kaldt jeres variable noget andet.

Læreren, som er ansvarlig for det frivillige arbejde indser imidlertidigt at du kun har en bestemt mægnde tid til hvert job, her 3 timer. Hun beslutter sig for at estimere hvor lang tid det tager for hvert barn er udføre hvert job, hvilket kan ses nedenfor og angiver $\text{TidKrævet}_{j,c} \in \{1, 2, 3, 4, 5\}$.

		Barn				
		c_1	c_2	c_3	c_4	c_5
	j_1	1	2	1	4	4
	j_2	6	2	4	2	2
Job	j_3	3	3	2	4	4
	j_4	1	1	4	4	2
	j_5	7	2	2	3	1

eller mere kopieringsvenligt

```
TidKrævet =[
    1 2 1 4 4;
    6 2 4 2 2;
    3 3 2 4 4;
    1 1 4 4 2;
    7 2 2 3 1
]
```

c. Implementér den opdaterede bibetingelse, og løs problemet. Prøv at kør kommandoen `println("x:  ",value.(x))` for din variable x. Hvad ser du?

d. Tilføj `Bin` som bibetingelse, når du definerer variablene (hvis du ikke allerede har gjort det). Løs problemet igen. Tjek om du får en mulig løsning.

5.3 Hjælp virksomheden med at minimere transportomkostninger

En virksomhed har 4 produktionssteder og 4 forskellige grossister for sit produkt.Virksomheden ønsker at minimere transportomkostningerne ved leverancer fra fabrik til grossist. Stykomkostningerne ved alternative leveringsmuligheder er som vist:

Der produceres tolv enheder i fabrikkerne A og B, otte enheder i fabrikkerne C og D. Grossisterne 1 og 2 skal have leveret henholdsvis ni og elleve enheder, mens 3 og 4 begge skal have ti enheder.

		Grossister				Beholdning
		G1	**G2**	**G3**	**G4**	
Fabrikker	A	2	4	10	9	12
	B	4	1	12	10	12
	C	10	8	2	4	8
	D	9	9	2	2	8
Efterspørgsel		**9**	**11**	**10**	**10**	

a. Find den optimale transportplan.

Virksomheden opnår nu rabat på transporten fra fabrik B til grossist 3, så at stykprisen ændres fra tolv til ni.

b. Hvordan påvirker det den optimale plan?

Efterspørgslen fra grossist 1 vokser nu fra ni til tretten.

c. Hvad bliver det optimale transportmønster, hvis det ikke er muligt at øge produktionen?

Det besluttes nu, at de fire yderligere enheder, som grossist 1 efterspørger, skal produceres fabrikkerne (A,B,C,D) således, at de samlede transportomkostninger minimeres.

d. Hvordan skal der nu produceres og transporteres?

5.4 Groceries

Virksomheden *Groceries* har 6 kunder, og skal levere dagligvarer med en vogn til sine kunder. Virksomheden vil gerne køre så kort som muligt, og virksomheden opstiller derfor to begrænsninger for ruten:

i. Man må kun køre ind i hver by én gang.

ii. Man må kun forlade hver by én gang.

Du får her givet koordinaterne til kunderne, noget af koden for at beregne afstandene mellem kunderne og kode til at definere variable til 0:

```
# PARAMETERS
C=6 # no of customers

coord = zeros(C,2)
coord[1,1]=0; coord[1,2]=0
coord[2,1]=104; coord[2,2]=19
coord[3,1]=370; coord[3,2]=305
coord[4,1]=651; coord[4,2]=221
coord[5,1]=112; coord[5,2]=121
coord[6,1]=134; coord[6,2]=515
```

```
Distance = zeros(Float64,C,C)
for c1 in 1:C
    for c2 in 1:C
        Distance[c1,c2]= sqrt(  #### Fyld et udtryk ind her ####)
    end
end

# MODEL
TSP = Model(HiGHS.Optimizer)

@variable(TSP, x[1:C,1:C],Bin)
for c1 in 1:C
    fix(x[c1,c1],0; force = true)
end
```

a. Implementer problemet og løs det i Julia.

Prøv at køre følgende kode til sidst:

```
println("objective = $(objective_value(TSP))")
println("Solve time: $(solve_time(TSP))")
println(round.(Int8,value.(x)))
```

b. Identificér problemet for virksomheden, og implementér nu et Travelling Salesman Problem. Kør koden ovenfor igen, og tjek at problemet er løst og tjek hvad der sker med køretiden af programmet.

Hint: Ved at installere pakken `Combinatorics`, kan du implementere definitionen fra forelæsningerne direkte:

```
using Combinatorics
for S in 2:C-1  # Iterating over the size of the subsets
    for subset in combinations(1:C, S)
        @constraint(TSP, sum(x[i,j] for i in subset for j in subset) <= S - 1)
    end
end
```

Alternativt, kan du bruge den simplere begrænsning, der ikke kræver nye pakker, og er ækvivalent, men formuleret anderledes.

```
@variable(TSP, u[1:C] >= 0)
# counter constraint
@constraint(TSP,
            counter_con[c1=1:C,c2=2:C,c1!=c2],
            u[c1] + 1 <= u[c2] + C*(1-x[c1,c2])
)
```

Ingen af de to tilgange er særlig effektive i praksis - men du kan prøve at tjekke hvilken, som er bedst her.

c. Undersøg køretiden af programmet når problemet stiger i kompleksitet.

Lineær Programmering (Opgave 1, Eksamen 2022) - 40%

Se ugeseddel 2. Du burde nu være i stand til at løse alle opgaver med Julia - og dermed 40% af eksamen.