(SMM641) - Revenue Management & Pricing. Network Revenue Management Part 1 - R Supplement

Oben Ceryan

Contents

1 Network Revenue Management with Dynamic Programming

1.1 Setting up the Dynamic Programming Algorithm

For details and formulation, please see slides.

```r
# Network Revenue Management with Dynamic Porgramming

N1=100; # Leg 1 seat availability
N2=120; # Leg 2 seat availability
TT=300; # Length of time horizon

arrivalprob=c(1/5, 4/15, 1/6, 4/15);

price=c(150,120,250,180);

# R requires arrays be created at the beginning.
# Creating empty arrays of correct dimensions.
# For the value function v(x1,x2,t):
v=array(rep( 0, len=(N1+1)*(N2+1)*(TT+1)), dim=c(N1+1,N2+1,TT+1));
# To keep track of optimal decisions,
# e.g. acceptance decision for a product 1 arrival: accept1(x1,x2,t):
accept1=array(rep( 0, len=(N1+1)*(N2+1)*(TT+1)), dim=c(N1+1,N2+1,TT+1));
# remaining are similarly defined and created.
accept2=array(rep( 0, len=(N1+1)*(N2+1)*(TT+1)), dim=c(N1+1,N2+1,TT+1));
accept3=array(rep( 0, len=(N1+1)*(N2+1)*(TT+1)), dim=c(N1+1,N2+1,TT+1));
accept4=array(rep( 0, len=(N1+1)*(N2+1)*(TT+1)), dim=c(N1+1,N2+1,TT+1));

totalarrivalprob=sum(arrivalprob);
noarrivalprob=1-totalarrivalprob;

# Terminal Values
for(i in 1:(N1+1)){
    for(j in 1:(N2+1)){
        v[i,j,1]=0; # All seats worthless at the end of horizon, i.e., t=1.
    }
}

# Dynamic Programming Algorithm
```

```r
for(t in 2:(TT+1)){ #2:TT+1
    for(i in 1:(N1+1)){ #1:N1+1
        for(j in 1:(N2+1)){ #1:N2+1

            # For no arrivals:
            vforarrival0=v[i,j,t-1];

            # For Product 1 arrival:
            # default not accept unless able/profitable to accept
            vforarrival1=v[i,j,t-1];
            accept1[i,j,t]=0;
            # If resource available:
            if(i>1){
                vforarrival1=max(price[1]+v[i-1,j,t-1],v[i,j,t-1]);
                # Recording the decision in the accept1 variable:
                if(price[1]+v[i-1,j,t-1]>v[i,j,t-1]){
                    accept1[i,j,t]=1;
                }
            }

            # For Product 2 arrival:
            # default not accept unless able/profitable to accept
            vforarrival2=v[i,j,t-1];
            accept2[i,j,t]=0;
            # If resource available:
            if(j>1){
                vforarrival2=max(price[2]+v[i,j-1,t-1],v[i,j,t-1]);
                # Recording the decision in the accept2 variable:
                if(price[2]+v[i,j-1,t-1]>v[i,j,t-1]){
                    accept2[i,j,t]=1;
                }
            }

            # For Product 3 arrival:
            # default not accept unless able/profitable to accept
            vforarrival3=v[i,j,t-1];
            accept3[i,j,t]=0;
            # If resources available:
            if(i>1){
```

```
                if(j>1){
                    vforarrival3=max(price[3]+v[i-1,j-1,t-1],v[i,j,t-1]);
                    # Recording the decision in the accept3 variable:
                    if(price[3]+v[i-1,j-1,t-1]>v[i,j,t-1]){
                        accept3[i,j,t]=1;
                    }
                }
            }

            # For Product 4 arrival:
            # default not accept unless able/profitable to accept
            vforarrival4=v[i,j,t-1];
            accept4[i,j,t]=0;
            # If resources available:
            if(i>1){
                if(j>1){
                    vforarrival4=max(price[4]+v[i-1,j-1,t-1],v[i,j,t-1]);
                    # Recording the decision in the accept4 variable:
                    if(price[4]+v[i-1,j-1,t-1]>v[i,j,t-1]){
                        accept4[i,j,t]=1;
                    }
                }
            }

            # Obtaining the overall value function from its parts:
            v[i,j,t]=noarrivalprob*vforarrival0+
                arrivalprob[1]*vforarrival1+
                arrivalprob[2]*vforarrival2+
                arrivalprob[3]*vforarrival3+
                arrivalprob[4]*vforarrival4;
        }
    }
}
```

1.2 Maximum Revenue for Given Capacity Levels and Remaining Time

```
# Optimal Revenue Starting at
# Leg 1 seat availability: N1=100 and
```

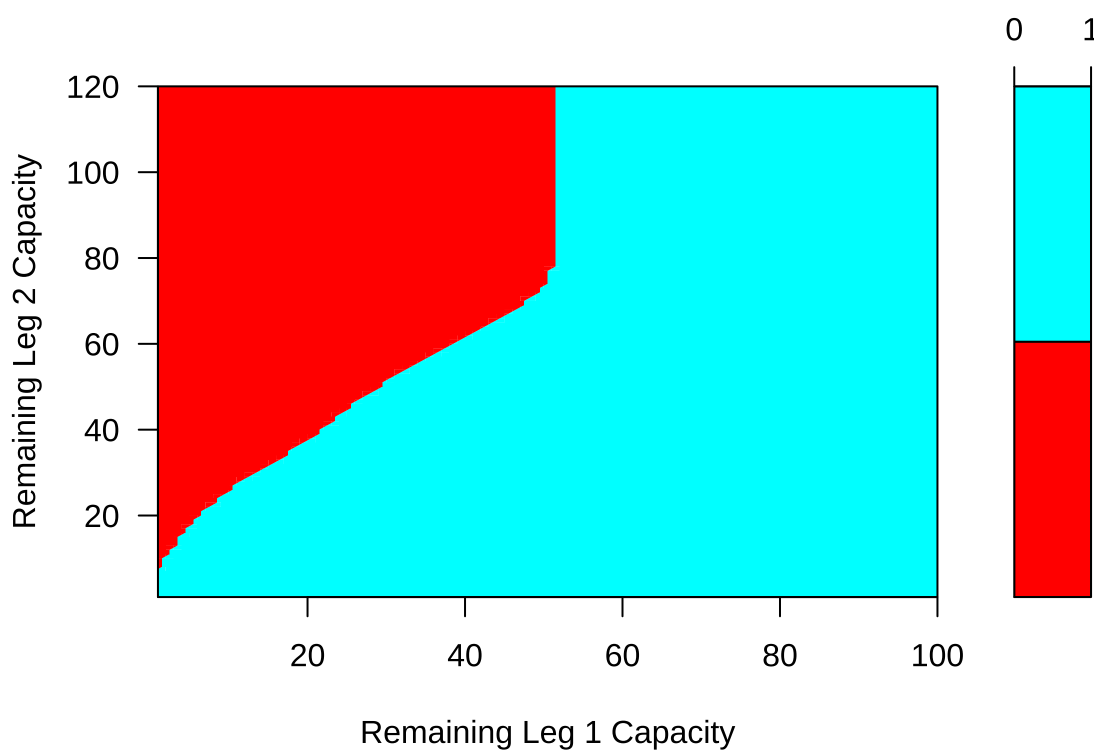
```r
# Leg 2 seat availability: N2=120
# with T=300 periods to go:
revenueDP<-v[101,121,301]
print(revenueDP)
```

```
## [1] 28375.28
```

2 Visualizing the Optimal Policy from Dynamic Programming

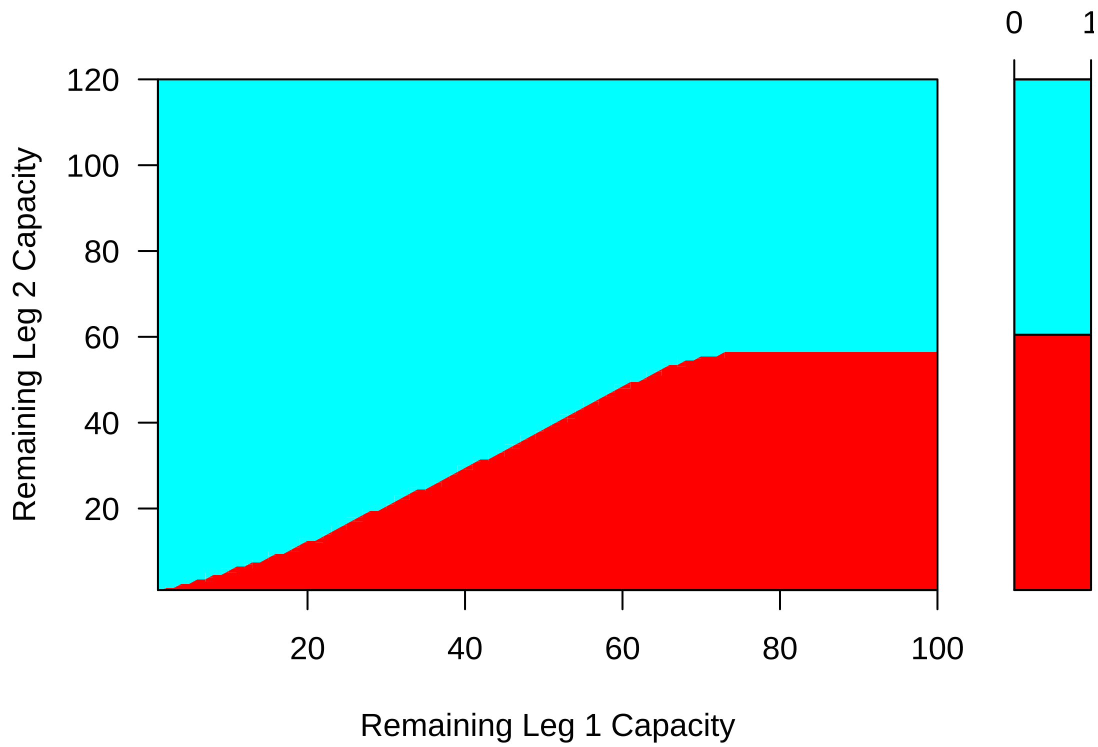
2.1 Optimal Allocation Policy for Product 1 with t=100 periods remaining

```r
acceptance<-accept1[2:101,2:121,101];
xaxis<-1:N1
yaxis<-1:N2
filled.contour(xaxis,yaxis,acceptance,xaxt="n",yaxt="n",
               zlim = range(acceptance, finite = TRUE),
               key.axes = axis(3, seq(0, 1, by = 1)),
               nlevels = 2,color.palette = rainbow,
               xlab="Remaining Leg 1 Capacity",
               ylab="Remaining Leg 2 Capacity")
```



2.2 Optimal Allocation Policy for Product 2 with t=100 periods remaining

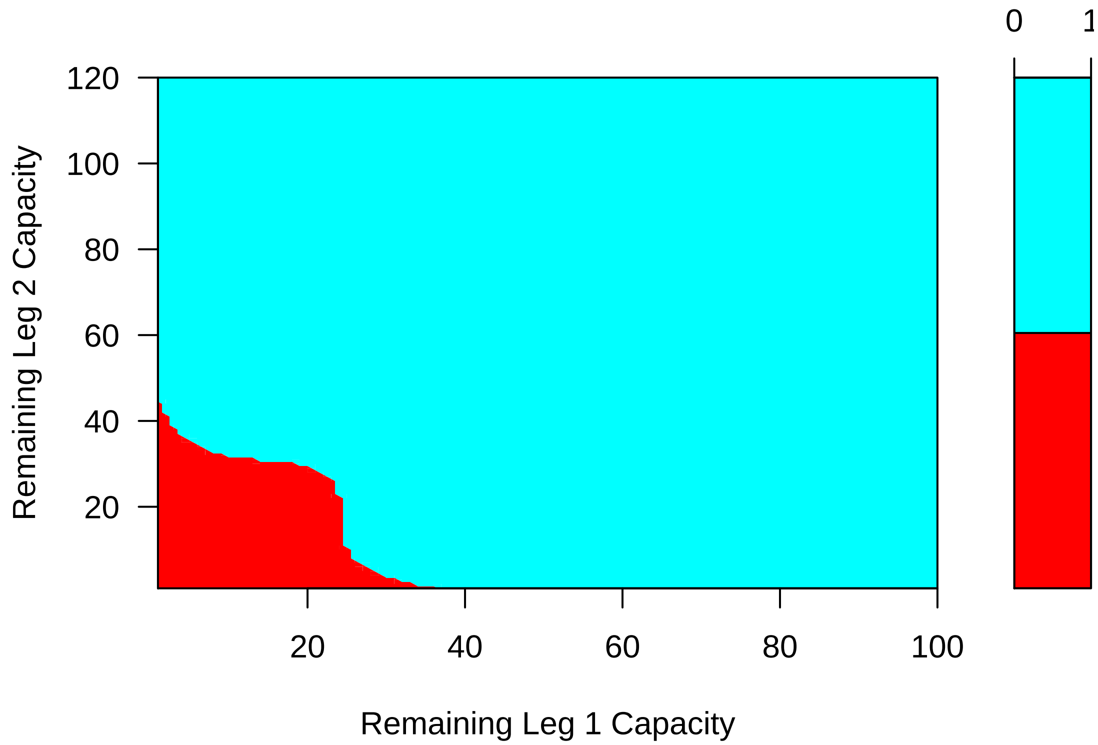
```
acceptance<-accept2[2:101,2:121,101];
xaxis<-1:N1
yaxis<-1:N2
filled.contour(xaxis,yaxis,acceptance,xaxt="n",yaxt="n",
               zlim = range(acceptance, finite = TRUE),
               key.axes = axis(3, seq(0, 1, by = 1)),
               nlevels = 2,color.palette = rainbow,
               xlab="Remaining Leg 1 Capacity",
               ylab="Remaining Leg 2 Capacity")
```



2.3 Optimal Allocation Policy for Product 3 with t=100 periods remaining

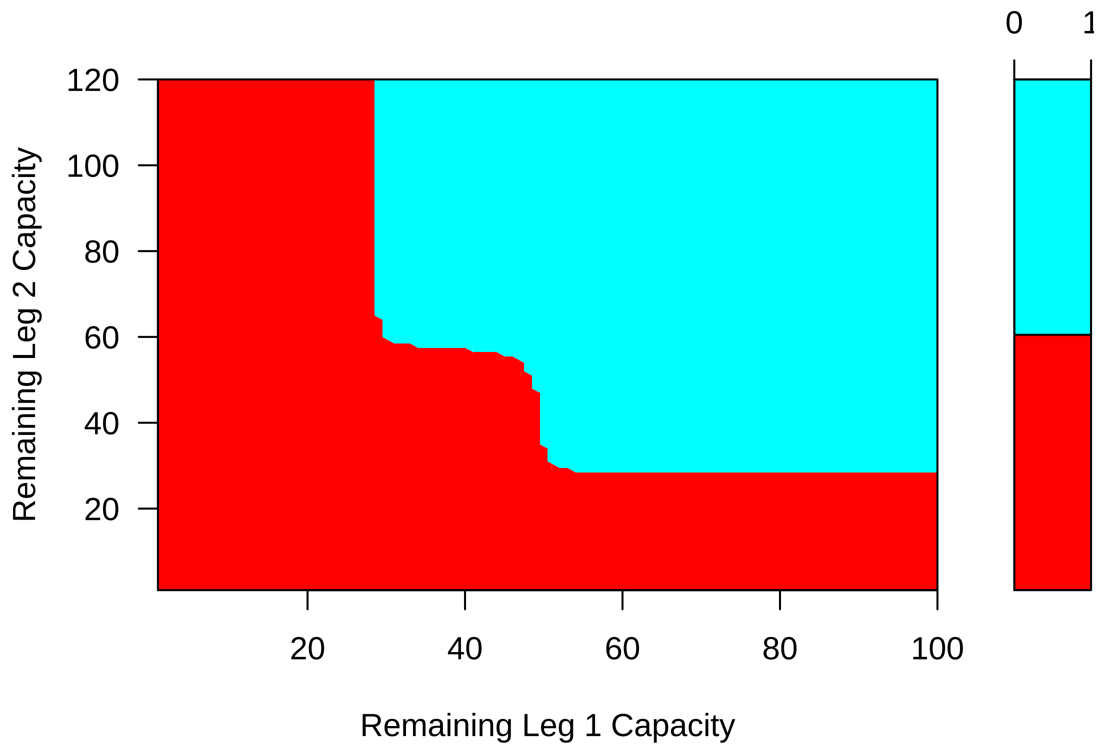
```
acceptance<-accept3[2:101,2:121,101];
xaxis<-1:N1
yaxis<-1:N2
filled.contour(xaxis,yaxis,acceptance,xaxt="n",yaxt="n",
               zlim = range(acceptance, finite = TRUE),
               key.axes = axis(3, seq(0, 1, by = 1)),
               nlevels = 2,color.palette = rainbow,
```

```
                xlab="Remaining Leg 1 Capacity",
                ylab="Remaining Leg 2 Capacity")
```



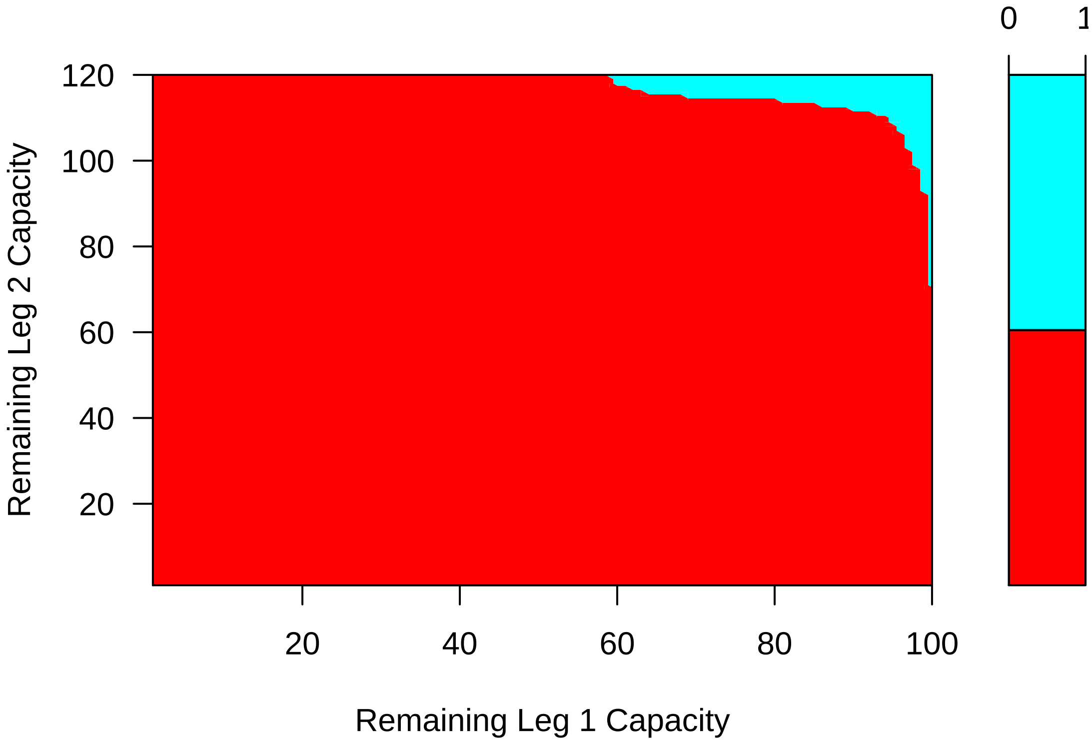
2.4 Optimal Allocation Policy for Product 4 with t=100 periods remaining

```
acceptance<-accept4[2:101,2:121,101];
xaxis<-1:N1
yaxis<-1:N2
filled.contour(xaxis,yaxis,acceptance,xaxt="n",yaxt="n",
               zlim = range(acceptance, finite = TRUE),
               key.axes = axis(3, seq(0, 1, by = 1)),
               nlevels = 2,color.palette = rainbow,
               xlab="Remaining Leg 1 Capacity",
               ylab="Remaining Leg 2 Capacity")
```



2.5 Optimal Allocation Policy for Product 4 with t=200 periods remaining

```
acceptance<-accept4[2:101,2:121,201];
xaxis<-1:N1
yaxis<-1:N2
filled.contour(xaxis,yaxis,acceptance,xaxt="n",yaxt="n",
               zlim = range(acceptance, finite = TRUE),
               key.axes = axis(3, seq(0, 1, by = 1)),
               nlevels = 2,color.palette = rainbow,
               xlab="Remaining Leg 1 Capacity",
               ylab="Remaining Leg 2 Capacity")
```


0
1
120
100
80
60
40
20
Remaining Leg 2 Capacity
20
40
60
80
100
Remaining Leg 1 Capacity

3 A First-Come First-Serve (FCFS) Allocation as a Lower Bound

```
# FCFS
# Below creating empty arrays
v=array(rep( 0, len=(N1+1)*(N2+1)*(TT+1)), dim=c(N1+1,N2+1,TT+1));
accept1=array(rep( 0, len=(N1+1)*(N2+1)*(TT+1)), dim=c(N1+1,N2+1,TT+1));
accept2=array(rep( 0, len=(N1+1)*(N2+1)*(TT+1)), dim=c(N1+1,N2+1,TT+1));
accept3=array(rep( 0, len=(N1+1)*(N2+1)*(TT+1)), dim=c(N1+1,N2+1,TT+1));
accept4=array(rep( 0, len=(N1+1)*(N2+1)*(TT+1)), dim=c(N1+1,N2+1,TT+1));

# Terminal Values
for(i in 1:(N1+1)){
    for(j in 1:(N2+1)){
        v[i,j,1]=0;
    }
}

# Dynamic Programming Recursion

for(t in 2:(TT+1)){ #2:TT+1
    for(i in 1:(N1+1)){ #1:N1+1
        for(j in 1:(N2+1)){ #1:N2+1

            # For no arrivals:
            vforarrival0=v[i,j,t-1];

            # For Product 1 arrival:
            # default not accept unless able to accept
            vforarrival1=v[i,j,t-1];
            accept1[i,j,t]=0;
            # If resource available:
            if(i>1){
                vforarrival1=price[1]+v[i-1,j,t-1];
                accept1[i,j,t]=1;
            }

            # For Product 2 arrival:
            # default not accept unless able to accept
            vforarrival2=v[i,j,t-1];
```

```
        accept2[i,j,t]=0;
        # If resource available:
        if(j>1){
            vforarrival2=price[2]+v[i,j-1,t-1];
            accept2[i,j,t]=1;
        }

        # For Product 3 arrival:
        # default not accept unless able to accept
        vforarrival3=v[i,j,t-1];
        accept3[i,j,t]=0;
        # If resources available:
        if(i>1){
            if(j>1){
                vforarrival3=price[3]+v[i-1,j-1,t-1];
                accept3[i,j,t]=1;
            }
        }

        # For Product 4 arrival:
        # default not accept unless able to accept
        vforarrival4=v[i,j,t-1];
        accept4[i,j,t]=0;
        # If resources available:
        if(i>1){
            if(j>1){
                vforarrival4=price[4]+v[i-1,j-1,t-1];
                accept4[i,j,t]=1;
            }
        }

        # Obtaining the overall value function from its parts:
        v[i,j,t]=noarrivalprob*vforarrival0+
            arrivalprob[1]*vforarrival1+
            arrivalprob[2]*vforarrival2+
            arrivalprob[3]*vforarrival3+
            arrivalprob[4]*vforarrival4;
    }
}
```

```r
}

# FCFS Revenue
revenueFCFS<-v[101,121,301]
print(revenueFCFS)
```

```
## [1] 25112.55
```