Das stp-Paket

Andreas Mors

Universität Augsburg
Andreas.Mors@gmail.com
9. November 2011

Inhaltsverzeichnis

1 Einleitung und mathematischer Hintergrund

Der folgende Bericht erklärt die Funktionsweise und Anwendung des R-Pakets `stp`. Mit diesem Paket lassen sich in R sogenannte Sitztransferpfade berechnen. Diese entstehen bei der Anwedung von zum Beispiel stationärer Divisormethoden auf Zuteilungsprobleme mit Stimmenvektor v ($v_i \geq v_j$ für $i < j$) und Hausgröße h. Bei gegebenem Stationaritätsparameter q lässt sich mittels der zugehörigen stationären Divisormethode eine Zuteilung A_q berechnen. Wenn man nun das gegebene q verändert, so ändert sich nicht zwangsläufig auch die Sitzzuteilung. Genauer gesagt liegt jedes q in einem abgeschlossenen Intervall I, indem für jedes $\bar{q} \in I$ die Zuteilung A_q eine mögliche Zuteilung ist. Startet man mit $q_1 = 0$ und der zugehörigen Zuteilung A_{q_1}, was $DivAuf$ entspricht, so kann man das Intervall $I = [a, b]$ für den Stationaritätsparameter q_1 berechnen, in dem sich die Sitzzuteilung nicht verändert. Die untere Grenze a ist in diesem Fall 0 und um die obere Grenze zu erhalten, müssen zunächst eine Reihe von Werten

$$q_{A_q}(i,j) = \frac{m_i^{A_q} v_j - (m_j^{A_q} - 1) v_i}{v_i - v_j}$$

berechnet werden. Dies muss für jedes Paar (i, j) mit $i < j$ geschehen. Sei hierbei $m_i^{A_q}$ die Anzahl der Sitze von Partei i in A_q, sowie v_i die Anzahl der Stimmen von Partei i, dann ist $q_{A_q}(i, j)$ der Parameter, an dem eine Bindung zwischen Partei i und Partei j entsteht. Das bedeutet, wenn dieser Parameter überschritten wird, gibt die Partei j einen Sitz an die Partei i ab, ein sogenannter Sitztransfer. Um die obere Grenze des Intervalls zu bestimmen, muss nun offensichtlich der kleinste aller eben berechneten Parameter gewählt werden. Bei diesem Parameter tritt zum ersten Mal ein Sitztransfer auf, davor bleibt die Zuteilung unverändert. Nun ist man wieder in der Ausgangssituation. Man hat eine (jetzt aktualisierte) Sitzzuteilung, welche sich von der vorherigen darin unterscheidet, dass Partei i einen Sitz von Partei j bekommen hat. Die untere Grenze des neuen Intervalls I ist die alte obere Grenze, und die neue obere Grenze berechnet sich analog zum ersten Fall. Wenn man diese Rechnung fortsetzt, landet man nach endlich vielen Schritten oder Transfers bei der Sitzzuteilung, die durch Abrunden entsteht, also $q = 1$ und hat somit den Sitztransferpfad berechnet.
Analog dazu kann man die selbe Überlegung auch für die Divisorverfahren mit Potenzmittelrundung durchführen. Dabei bezeichnen wir den

Rundungsparameter als p und starten bei $p = -\infty$, was wieder dem Divisorverfahren mit Aufrundung entspricht. In diesem Fall lassen sich die $p(i, j)$ aber nicht so einfach berechnen wie im stationären Fall. Die gesuchten p's sind die Lösungen folgender Gleichungen:

$$\left(\frac{m_i^p + (m_i + 1)^p}{(m_j - 1)^p + m_j^p}\right)^{\frac{1}{p}} = \frac{v_i}{v_j}$$

Man sieht leicht, dass diese Gleichung für $\frac{m_i+1}{m_j} > \frac{v_i}{v_j}$ keine Lösung besitzt, da die linke Seite in p monoton fallend ist und als Grenzwert für $p \to \infty$ $\frac{m_i+1}{m_j}$ besitzt. Analog gilt das selbe für den Grenzwert $p \to -\infty$ nämlich $\frac{m_i}{m_j-1}$, falls dieser echt kleiner als $\frac{v_i}{v_j}$ ist. Dies bedeutet, dass es für die aktuelle Sitzzuteilung keinen Parameter p gibt, welcher einen Sitztransfer von Partei i nach Partei j verursachen würde. Dies kann sich allerdings im späteren Verlauf des Pfads wieder ändern, da sich die m_i verändern können. Wenn es für kein Paar (i, j) eine Lösung der Gleichung gibt, ist die Zuteilung gleich der Zuteilung, die durch $p = \infty$ (*DivAbr*) entsteht. Im folgenden Teil, wird die Funktionsweise und Anwendung des `stp`-Pakets besprochen. Zur Benutzung des `stp`-Pakets werden die Zusatzpakete *RBazi* und *Brobdingnag* benötigt.

2 Funktionen

2.1 seatTransferPath(),stp()

2.1.1 Signatur

```
stp(votedata, hs=NULL, method='stat', point=TRUE, plin=FALSE,
jit=FALSE, out=3)
```

2.1.2 Argumente

votedata: In dem Objekt *votedata* werden die Stimmdaten übergeben. Das können entweder ein einzelner Stimmvektor oder mehrere Stimmvektoren sein. Im ersten Fall können die Stimmen als einfacher Vektor, als (n,1)-Matrix, als rBazi-Objekt oder als rBaziApportionment-Objekt übergeben werden. Mehrere Stimmvektoren werden als Matrix übergeben, welche die einzelnen Vektoren in den Spalten enthält.

hs [numeric]: In dem Objekt *hs* wird die Hausgröße übergeben. Falls der Funktion ein rBazi-Objekt oder ein rBaziApportionment-Objekt übergeben wird, muss die Hausgröße nicht angegeben werden, da diese in diesen Objekten bereits enthalten ist.

method [character]: In dem Objekt *method* wird der Typ der Divisormethode angegeben. Es kann hier durch 'stat' oder 'pot' zwischen einer Divisormethode mit stationärer Rundung oder einer Divisormethode mit Potenzmittelrundung gewählt werden.

out [integer]: Mit dem Objekt *out* wird die Ausgabegenauigkeit der Rundungsparameter in der Pfadtabelle spezifiziert. *out* gibt dabei die Anzahl der Nachkommastellen an.

Die folgenden Argumente sind nur interessant, falls mehrere Stimmvektoren übergeben werden, denn sie modifizieren die Graphik, die in diesem Fall erstellt wird.

point [logical]: Falls *point* `TRUE` ist, werden in dem Plot die einzelnen Punkte gezeichnet. Dieser Parameter ist nützlich, falls *plin* `TRUE` ist, dann könnte *point* = `FALSE` interessant sein.

plin[logical]: Falls *plin* `TRUE` ist, werden die Punkte, die zu einem Stimmvektor gehören mit Linien verbunden.

jit [logical]: Falls *jit* `TRUE` ist, werden die Punkte im Plot mit Jittering versehen.

2.1.3 Funktionsweise & Output

Die Funktion `stp()` ist die Hauptfunktion des Pakets, mit ihr kann man den Sitztransferpfad berechnen. Zunächst filtert sie aus den Eingabedaten die relevanten Informationen und überprüft, ob alle benötigen Objekte vorhanden sind. Dann werden in einer Schleife nach der Methode, die in der Einleitung beschrieben wurde, nach und nach die Sitztransfers berechnet. Der Rundungsparameter wird hierbei von der Funktion `calcq()` bestimmt. Dies geschieht so lange, bis die aktuelle Sitzzuteilung, der von *DivAuf* entspricht.
Falls nur ein Stimmvektor übergeben worden ist, gibt die Funktion eine

Objekt vom Typ *seattransferobjekt* zurück. Ein möglicher Output sieht dann folgendermaßen aus und ist im Prinzip selbsterklärend. Der zugehörige Plot zeigt den Verlauf der Sitze der einzelnen Parteien innerhalb des Pfades.

```
Sitztransferpfad mit 7 Transfers

         0.155 0.437 0.445 0.486 0.727 0.947 0.99  1
P1 42919    41    42    42    43    43    44   44 45
P2 13048    13    13    13    13    13    13   13 13
P3 10879    11    11    11    11    11    11   11 11
P4 10581    10    10    11    11    11    11   11 11
P5 9547     10     9     9     9    10     9   10 10
P6 5708      6     6     6     6     6     6    5  5
P7 2502      3     3     3     3     2     2    2  2
P8 1898      2     2     2     2     2     2    2  1
P9 1461      2     2     2     1     1     1    1  1
P10 1457     2     2     1     1     1     1    1  1


Liste der Sitztransfers:

                ST1 ST2 ST3 ST4 ST5 ST6 ST7
Sitz von Partei   5  10   9   7   5   6   8
zu Partei         1   4   1   5   1   5   1

Genaue Liste der qs:

    q
ST0 "0.154500779096248"
ST1 "0.437198597106532"
ST2 "0.444859858169714"
ST3 "0.486018452803407"
ST4 "0.726657077789764"
ST5 "0.94738213076322"
ST6 "0.989566319689915"
ST7 "1"
```

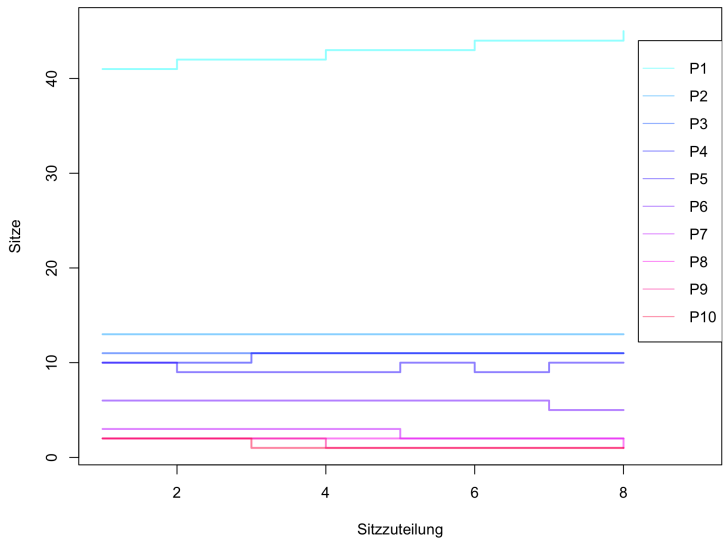

Abbildung 1: Output des Befehls `plot(<<stp-Objekt>>)`

Der Output für den Fall, dass mehrere Stimmvektoren übergeben worden sind, ist etwas komplizierter. Das Problem ist hierbei, dass bei n verschiedenen Stimmvektoren, n verschiedene Pfade betrachtet werden müssen, welche zudem unterschiedlich lang sein können. Der Output besteht aus zwei Grafiken und versucht folgende Sachverhalte darzustellen:

1. Wie oft gab es wie viele Sitztransfers?
2. Wie sind die Sitze beim jeweils i-ten Transfer gewandert?
3. Wie sind die Transfers innerhalb des i-ten Sitztransfer verteilt?
4. Wie ist der Verlauf der Transfers innerhalb eines Pfades?
5. Wie sind die Rundungsparameter insgesamt verteilt?

Dies geschieht abhängig von der Anzahl n der Stimmvektoren auf zwei unterschiedliche Arten und Weisen (per Defaulteinstellung). Ein Output für $n < 20$ sieht man in Abbildung 2.
Besprechen wir zunächst die einzelnen Elemente in einem Quadrat. Der obere Balken gibt an, wie viel Prozent der gesamten Pfade in diesem

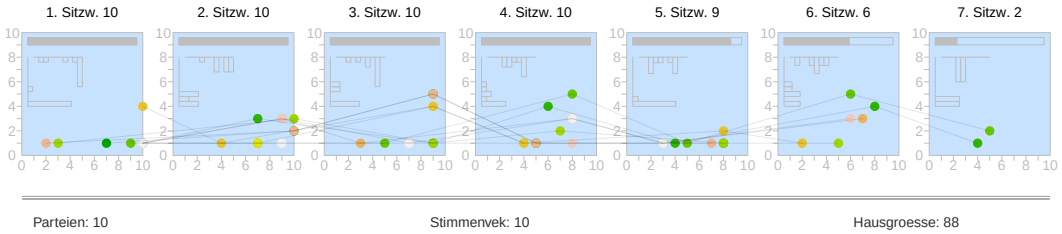

Abbildung 2: Output für $n < 20$ Teil 1

Quadrat dargestellt sind (Die exakte Anzahl steht am Ende der jeweiligen Überschrift). Die einzelnen Punkte stellen die Sitztransfers dar. Im ersten Quadrat bedeutet somit der dunkelgrüne Punkt, dass beim zugehörigen Stimmvektor der erste Sitz von Partei 7 an Partei 1 transferiert wurde. Wenn man der Linie folgt, sieht man dass in diesem Pfad der zweite Sitz von Partei 7 an Partei 3 ging, dann von 9 an 1, von 6 an 4, von 4 an 1, von 8 an 4 und schließlich von 4 an 1. Die zwei Histogramme zeigen an, wie die Parteien die einen Sitz bekommen haben (vertikal) und die Parteien die einen Sitz abgeben mussten (horizontal) jeweils beim i-ten Transfer verteilt sind. Aus der zweiten Graphik des Outputs (Abbildung 4) kann man ablesen, wie die Rundungsparameter insgesamt verteilt sind.
Wenn man sehr viele verschiedene Stimmvektoren hat, wird es mit den verschiedenen Farben sehr unübersichtlich, deswegen werden die Punkte dann alle in schwarz, mit angepasstem alphablending geplottet. Einen solchen Output sieht man in der folgenden Abbildung und Abbildung 5.

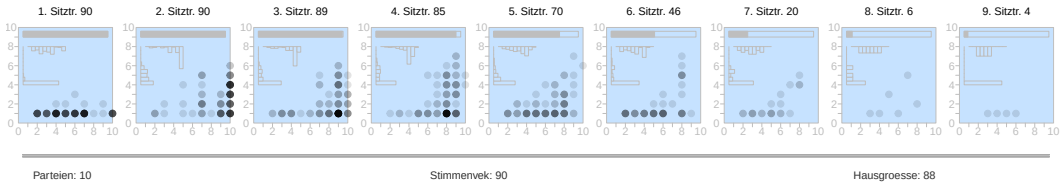

Abbildung 3: Output für $n > 20$ Teil 1

In diesem Fall wurde mit Potenzmittelrundung gerechnet und im zugehörigen Histogramm sieht man, dass die Rundungsparameter nahezu symmetrisch um 0 verteilt sind.

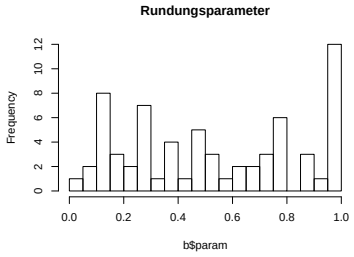

Abbildung 4: Output für $n < 20$ Teil 2

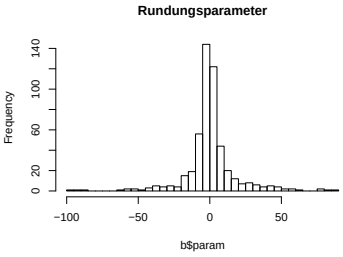

Abbildung 5: Output für $n > 20$ Teil 2

2.2 calcq()

2.2.1 Signatur

```
calcq(seats,votedata,method='stat')
```

2.2.2 Argumente

seats [numeric]: In dem Objekt *seats* wird die aktuelle Sitzzuteilung übergeben.

votedata [numeric]: *votedata* enthält den Stimmvektor.

method [character]: In dem Objekt *method* wird der Typ der Divisormethode angegeben. Es kann hier durch 'stat' oder 'pot' zwischen einer Divisormethode mit stationärer Rundung oder einer Divisormethode mit Potenzmittelrundung gewählt werden.

2.2.3 Funktionsweise

Wie in Abschnitt 2.1 schon beschrieben wurde, wird mit `calcq()` der Rundungsparameter berechnet, bei dem der nächste Transfer stattfindet. Falls mit stationärer Rundung gerechnet wird, ist das ganze problemlos. Es werden - wie in der Einleitung beschrieben - alle Parameter für die Paare von Parteien (i, j) mit $i < j$ berechnet und der kleinste zurückgegeben. Im anderen Fall ist das etwas komplizierter, denn die Gleichungen besitzen nicht immer eine Lösung, was dann bedeutet, dass es für dieses Paar mit dieser Sitzzuteilung für kein Parameter p zu einem Transfer kommen wird. (Im stationären Fall würde das bedeuten,

dass der Parameter größer als 1 ist) Aus diesem Grund berechnet die Funktion zunächst die beiden Grenzwerte für $p \to \pm\infty$ und überprüft, ob überhaupt eine Lösung existiert. Falls eine existiert, tritt das nächste Problem auf. Numerisch wird versucht von der Funktion

$$fpot(m_i, m_j, v_i, v_j) = \left(\frac{m_i^p + (m_i+1)^p}{(m_j-1)^p + m_j^p}\right)^{\frac{1}{p}} - \frac{v_i}{v_j}$$

eine Nullstelle zu suchen. Da das eine monoton fallende Funktion ist, kann das per Intervallschachtelung geschehen. Dazu müssen oft Funktionswerte dieser Funktion ausgewertet werden und genau das ist das Problem. Für Sitzzuteilungsprobleme mit großer Hausgröße werden die Terme im Nenner und Zähler größer, als es mit normaler Fließpunktrechnung realisierbar wäre. Deswegen wird auf das Zusatz-R-Paket *Brobdingnag* zurückgegriffen, welches mit wesentlich größeren Zahlen rechnen kann. Das führt pro Funktionsaufruf zu einem zusätzlichen Zeitaufwand von einem Faktor größer 1000. Folglich muss man den Einsatz dieses Pakets kontrollieren, dass die Laufzeiten der Funktion noch praktikabel sind. Dies wird hier so umgesetzt, dass vor jedem Funktionsaufruf überprüft wird, ob es zu einem „Überlauf“ kommt oder nicht und dann entsprechend mit normaler Genauigkeit oder mit hoher Genauigkeit gerechnet wird. Das verdoppelt zwar auch die Laufzeit der Funktion, ist aber insgesamt gesehen noch wesentlich schneller. Um die Berechnung noch schneller zu machen, wird zunächst eine Nullstelle im Intervall [-100,100] gesucht, falls diese nicht gefunden wird, wird das Intervall [-1e6,1e6] betrachtet. Alle praktisch vorkommenden Rundungsparameter sollten dadurch gefunden werden. Es ist auch möglich, dass der letzte Transfer erst im Grenzwert $p = \infty$ auftritt, dieser Fall wird vom Programm aber auch erkannt.

2.3 datagen()

2.3.1 Signatur

```
datagen(votes,n,dev)
```

2.3.2 Argumente

votes[numeric]: Der zu störende Stimmvektor.

n[integer]: Anzahl der zu erzeugenden Vektoren.

dev[numeric]: dev gibt an, wie groß die Standardabweichung der Störung ist. $sd_j = votes_j/dev$.

2.3.3 Funktionsweise

`datagen()` generiert ausgehend von dem Stimmvektor $votes$, insgesamt $n-1$ Stimmvektoren, indem jede Komponente j normalverteilt gestört wird mit $sd_j = votes_j/dev$ und Mittelwert 0. Der ursprüngliche Vektor wird auch mit zurückgegeben.

2.4 Zusatzfunktionen

Alle anderen im Paket enthaltene Funktionen sind für die Anwendung uninteressant, deswegen wird hier auf eine Erläuterung verzichtet. Abschließend noch ein paar Codebeispiele.

3 Beispiele

```
votes1<-c(42919,13048,10879,10581,9547,5708,2502,1898,1461,1457)
votes2<-c(27744,25178,19951,14610,9225,3292)
votes3<-c(11828277,9990488,6316080,5155933,4643272,2830238) #votes of the
#main apportionment of the DBT 2009
parteienbt<-c("CDU","SPD","FDP","LINKE","GRUENE","CSU")
votes4<-cbind(votes3)
rownames(votes4)<-parteienbt

#stationary
(a<-stp(votes1,100))
plot(a)
summary(a)
(b<-stp(votes4,598))

#power mean
(c<-stp(votes2,36,"pot"))
plot(c)

#muli
stp(datagen(votes1,250),88,jit=FALSE)
stp(datagen(votes1,10),88,jit=FALSE,plin=TRUE)
```

4 Quellenverzeichnis

Marshall AW, Olkin I, Pukelsheim F (2002) A majorization comparison of apportionment methods in proportional representation. Social Choice and Welfare 19: 885-900