Lösen von Gleichungssystemen und symbolische Gleichungen in R

Kolja Hopfmann
Betreuer: Eugen Betke
Informatik PIR16

08/06/2016

1 Einführung

Symbolische Mathematik ist heutzutage in der Informatik tief verwurzelt, da viele Anwendungsbereiche, wie zb. maschinelles Lernen, Signalverarbeitung und Laufzeitberechnung von ihr profitieren. Daraus folgt die Relevanz und Abhängigkeit diese Form der Mathematik auch in IT-Systeme einzubetten um somit den nutzen der Rechnertechnik und der symbolischen Mathematik miteinander zu kombinieren. Dadurch ist man in der Lage komplexe Funktionen und Gleichungen mit schneller Hardware effizienter zu verarbeiten als je zuvor. Obwohl die Programmiersprache R hauptsächlich in hinsicht auf Anwendungen im Bereich der Statistik entwickelt wurde, ist R durch die GNU-Lizenz im laufe der Zeit erweitert worden, sodass es mittlerweile möglich ist symbolische Mathematik in R zu verarbeiten.

1.1 Definition

CAS. *In Computeralgebrasystemen(CAS) bedeutet der Ausdruck symbolische Mathematik, dass Operationen und Kalkulationen von mathematischen Ausdrücken mit Variablen auf Computern ausgeführt werden.“*

[Planetmath.org]

Unter einem Computeralgebrasystem verstehen wir also ein Programm mit welchem wir in der Lage sind symbolische Mathematik durchzuführen. Um also, wie oben genannt, Hardwareleistung mit symbolischer Mathematik zu verknüpfen brauchen wir ein CAS. Hier stellt sich also nun die Frage ob R ein vollständiges Computeralgebrasystem ist.

1.2 Operationen

Um zu untersuchen man R als ein CAS verwenden kann, gilt es symbolische Mathematik auf einzelne fundamentale Operationen und Anwendungen zu differenzieren und versuchen diese in der R Umgebung zu finden:

1. Lösung eines Gleichungssystems(GLS) bilden.
2. Matrix-Operationen.
3. Arithmetische Operationen auf Polynomen.
4. Vereinfachen von komplizierten symbolischen Ausdrücken in eine Standardform.
5. Wertzuweisung der Variablen und Berechnung.
6. Ableiten von symbolischen Ausdrücken.
7. Bilden von Integralen.

Sind nun alle diese Themenbereiche der symbolischen Mathmatik über [CRAN] verfügbar, ist dies ein Argument dafür, dass R als ein grundlegendes Computeralgebrasystem behandelt werden kann.

1.3 Schnittstellen zu Algebrasystemen

Desweiteren existieren über [CRAN] auch Schnittstellen zu Vollständigen Computeralgebrasystemen mit denen man in der Lage ist, dass gewünschte CAS direkt in der R-Umgebung anzusprechen. Dazu später mehr.

2 Gleichungssysteme

Unter einem Gleichungssystem verstehen wir eine Menge von Gleichungen mit Unbekannten Symbolen die untereinander in Beziehung stehen. Dies ist relevant, da man so in der Lage ist, für bestimmte Prämissen die Werte der Symbole zu ermitteln.

2.1 solve

2.1.1 Vorstellung

Zum Lösen von Gleichungssystemen in R verwenden wir das Paket Solve, welches ermöglicht für ein gegebenes GLS in Matrixform die Unbekannten Variablen zu ermitteln, indem wir die solve-Funktion mit zwei Parametern aufrufen: Zunächst eine beliebige $n \times n$-Matrix als 1. Parameter und als 2. Parameter einen n-Vektor oder wieder eine $n \times n$-Matrix. Übergeben wir nur einen Parameter in Form einer Matrix wird diese Invertiert. Solve ist ein Build-In Paket von R und steht uns somit nach der Installation von R sofort zur Verfügung.

2.1.2 Beispiele

Sei nun folgendes GLS gegeben:

$$1x + 2y = 5$$
$$3x + 4y = 6$$

Daraus ergibt die die Matrixform.

$$\begin{bmatrix} 1 & 2 & 5 \\ 3 & 4 & 6 \end{bmatrix}$$

Nun erfolgt die Teilung in Matrix und Vektor für beide Seiten der Gleichungen.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 5 \\ 6 \end{bmatrix}$$

Nun erzeugen wir Matrix und Vektor in R und übergeben beides an die solve-Funktion und kriegen die gewünschte Ausabe.

```
>A = matrix(c(1,2,3,4),nrow=2,ncol=2,byrow=T)
>B  = c(5,6)
>solve(A,B)
[1] -4.0 4.5
```

Übergeben wir nur eine Matrix als Parameter so ist der 2. Parameter die n-te Einheitsmatrix I^n. Folgend ergeben die gesuchten Variblen genau die invertierte Matrx da folgendes gilt.

$$A \times A^{-1} = I$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times A^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Übersetzt in R-Code ergibt sich als als A^{-1}:

```
>A = matrix(c(1,2,3,4),nrow=2,ncol=2,byrow=T)
>solve(A)
     [,1] [,2]
[1,] -2.0  1.0
[2,]  1.5 -0.5
```

Ein Negativtest zeigt die Robustheit von Solve, indem wir folgendes GLS bilden:

$$1x + 2y = 5$$
$$0x + 0y = 4$$

In Matrixform ist deutlich zu erkennen, dass das GLS eine Nullzeile besitzt und somit nicht lösbar sein kann.

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & 0 & 4 \end{bmatrix}$$

Übergeben wir die Matrix nun als Parameter kriegen wir folgende Fehlermeldung.

```
>A = matrix(c(1,2,0,0),nrow=2,ncol=2,byrow=T)
>solve(A,c(3,4))
Error in solve.default(A, c(3, 4)) :
Lapack routine dgesv: system is exactly singular: U[2,2]=0
```

2.1.3 Vor-/Nachteile

Mit Solve besitzen wir nun die Fähigkeit Gleichungssysteme effezient über die R-Umgebung zu lösen und somit Fehler zu minimieren. Von Nachteil is die Notwendigkeit, dass wir Gleichungen als Matrix übergeben müssen und somit gezwungen sind diese von Hand umzuformen. Außerdem kann R nur direkte Werte für Variablen ausgeben. Die Angabe von Lösungsmengen ist nicht möglich, da einfach ein beliebiges Element aus der Menge verwendet wird.

3 Symbolische Manipulation

Unter symbolischer Manipulation verstehen wir den Umgang mit symbolischen Ausdrücken ohne Erwartung einer bestimmten Wertzuweisung der Variablen. Umgangssprachlich spricht man auch von "Rechnen mit Buchstaben".

3.1 mpoly

3.1.1 Vorstellung

Mpoly ist ein Paket für arithmetische Operationen mit Polynome, indem es einen neuen Objekttyp in R verwendet auf dem diverse Rechenoperationen zur Verfügung stehen. Man spricht die mpoly-Funktion an und übergebt das zu erzeugende Polynom als Parameter in Form eines einfachen Textstrings. Dazu sind wir in der Lage die Polynome anschließend in Funktionen umzuwandeln mit denen wir den abhängigen Symbolen Werte zuweisen können. Desweiteren sind wir später in der Lage diese dann in Abhängikeit einer Variable abzuleiten.

3.1.2 Beispiele

Wir wollen nun folgendes Polynom in R bearbeiten.

$$f = x^3 + x^2 + x + 1$$

Dies übergeben wir nun an die Funktion.

```
> f = mp("x^3+x^2+x+1")
>f
x^3 + x^2 + x + 1
```

Nun existiert das Polynom in Form eines neuen Typs auf dem wir nun Rechenoperationen durchführen können.

```
>f-f
0
>f+f
2 x^3 + 2 x^2 + 2 x + 2
>f*f
x^6 + 2 x^5 + 3 x^4 + 4 x^3 + 3 x^2 + 2 x + 1
```

Auch das Abfragen auf Gleichheit ist möglich: Im Folgenden vergleichen wir f mit sich selber und mit dem Mononom x, was jeweils $TRUE/FALSE$ liefert.

```
>f==f
[1] TRUE
>f == mp("x")
[1] FALSE
```

Neben eigenständigen mp-Objekten können wir auch Polynome mit dem Befehl mpolyList in Listen verwalten.

```
>f = mp("x^3+x^2+x+1")
>g = mp("x^2")
>h = mp("2y")
>k = mpolyList(f,g,h)
>k
x^3  +  x^2  +  x  +  1
x^2
2 y
```

Nun können wir Listen mit Operationen verknüpfen. Hierbei wird das i-te Element der 1. Liste mit dem i-ten Element der 2. Liste verknüpft.

```
>k+k
2 x^3 + 2 x^2 + 2 x + 2
2 x^2
4 y
>k-k
0
0
0
>k*k
x^6 + 2 x^5 + 3 x^4 + 4 x^3 + 3 x^2 + 2 x + 1
x^4
4 y^2
```

Da wir nun unsere Polynome ausreichend bearbeitet haben, wollen wir deren $f(x)$ Wert auslesen. mp-Objekte alleine, bieten diese Funktion jedoch nicht. Es ist vonnöten, dass wir die Polynome in auslesbare Funktionen umwandeln. Dies ist mit der as.funktion-Methode möglich.
Sei nun der Einfachheit nach folgende Funktion f gegeben.

$$f(x) = x^2$$
$$f(0) = 0$$
$$f(2) = 4$$

Wir übergeben nun an die as.function-Methode ein beliebiges mp-Objekt als Parameter und sind somit in der Lage f Werte zuzuweisen.

```
>f = as.function(mp(x^2))
f(.) with . = x
>f(0)
[1] 0
>f(2)
[1] 4
```

Eine weitere Option ist das ableiten von Funktionen, hier bilden wir zunächst wieder das Standartpolynom des dritten Grades und bilden dazu die erste Ableitung.

$$f(x) = x^3 + x^2 + x + 1$$
$$f'(x) = 3x^2 + 2x + 1$$

In R erstellen wir parallel das mp-Objekt und verwenden die deriv-Funktion mit dem Polynom und der abzuleitenden Variable als Parameter und gelangen somit zur gewünschten Ausgabe.

```
>f = mp("x^3+x^2+x+1")
>deriv(f, 'x')
3 x^2 + 2 x + 1
```

Weitere kleine Möglichkeiten zur Bequemlichkeit in mpoly bilden die Operationen totaldeg und mononomials, wobei totaldeg den Grad den Polynoms angibt und monomials die eintelnen Mononome ausgibt.

```
>f = mp("x^3+x^2+x+1")
>totaldeg(f)
[1] 3
>monomials(f)
x^3
x^2
x
1
```

3.1.3 Vor-/Nachteile

Mpoly deckt einen enorm großen Bereich der symbolischen Manipulation ab. Mit dem Paket sind wir in der Lage Polynome zu erstellen und diese zu bearbeiten. Der Vorteil is hier ein eigen neu definierter Typ für Polynome, was zur Verständlichkeit beiträgt. Grundlegendes zu Funktionen deckt mpoly, jedoch nur unzureichend ab. Zwar können wir Polynome in Funktionen umwandeln, doch ableiten können wir demnach auch nur Polynomialfunktionen. Später im Verlauf werden wir dies auf andere Funktionstypen ausweiten.

3.2 simplr

3.2.1 Vorstellung

Simplr ist ein Paket für R unter [CRAN] welches zur Vereinfachung von symbolischen Ausdrücken dient. Mit dem Befehl symplifyq können wir eine gewünschte R expression, welche einen symbolischen Ausdruck darstellt vereinfachen. Simplr geht immer nach demselmben Schema vor, demnach ergibt sich aus den durch simplr vereinfachten ausdrücken eine Standardform.

3.2.2 Beispiele

Im ersten Beispiel die offensichliche Regel, dass ein Koeffizient von 0 den Ausdruck eliminiert. Desweiteren kann simplifyq auf trigonometrische Ausdrücke vereinfachen. Zuletzt lässt sich die Standardform erkennen, die bei additiven Ausdrücken immer ein Monomom zu beginn beinhaltet, mit dem Rest geklammert.

```
>simplifyq((a+b)*0)
[1] 0
>simplifyq(sin(x)^2+cos(x)^2)
[1] 1
simplifyq(a+a+a+b+b+b+c+c+c)
[1] 3 * c + (3 * b + 3 * a)
```

3.2.3 Vor-/Nachteile

Simplr bietet mit der Operation simplifyq eine Methode symbolische Ausdrücke zu vereinfache, was einen weiteren Bereich der symbolischen Mathematik abdeckt. Simplr geht über ganzrationale Ausdrücke hinaus, mit der Möglichkeit trigonometrische und exponentielle Ausdrücke zu vereinfachen. Ein Nachteil ist, dass simplr nur R-expressions verarbeiten kann. Das heißt eine direkte Verknüpfung mit Paketen wie mpoly gibt es nicht.

3.3 D

3.3.1 Vorstellung

Mit der Build-In Funktion D besitzt R die Fähigkeit expressions einer vordefinierten Form abzuleiten.

3.3.2 Beispiele

Zu beginn erstellen wir den von der Syntax geforderten Ausdruck. Dieser besteht aus jeweils einem String, welcher den Ausdruck darstellt und einem character für die abhängige Variable. Dann geben wir die D Funktion an mit der expression und der abzuleitenden Variable.

$$f(x) = x^3 + x^2 + x + 1$$
$$f'(x) = 3x^2 + 2x + 1$$

```
>fx = expression("x^3+x^2+x+1", 'x')
>D(fx, 'x')
3 * x^2 + 2 * x + 1
```

Analog klappt dies auch mit trigonometrischen expressions.

$$h(x) = sin(x^2)$$
$$h'(x) = 2x * cos(x^2)$$

```
>hx = expression(sin(x^2),'x')
>D(hx,'x')
cos(x^2) * (2 * x)
```

D schließt auch Exponentialfunktionen mit ein. Bei der Funktion e^x erwarten wir als Ableitung wieder e^x selbst.

$$g(x) = e^x$$
$$g'(x) = e^x$$

```
>gx = expression(e^x,'x')
>D(gx,'x')
e^x * log(e)
```

3.3.3 Vor-/Nachteile

Nun können wir mit D auch über Polynomialfunktionen hinaus Ableitungen bilden. Dies ist ein enormer Vorteil gegenüber mpoly und relevant als Operation der symbolischen Mathematik. Jedoch bietet D keine weiteren Funktionen über Ableiten hinaus. Desweiteren können wir immernoch keine Integrale bilden, ein weiterer wichtiger Bestandteil für ein CAS.

3.4 Integrate

3.4.1 Vorstellung

Mit dem Paket Integrate kann man nun auch Integrale eindimensionaler Funktionen bestimmen. Inetgrate benutzt die Gauß-Quadratur als Annäherungsverfahren. [math.ethz.ch]

3.4.2 Beispiele

Wir definieren uns zunächst eine R-Funktion in Form eines symbolischen Ausdrucks mit einer abhängigen Variable als character und geben diese als unseren ersten Parameter an. Zwei weitere Parameter bilden die Intervallgrenzen. Optional lassen sich auch unbegrenzte Integrale definieren, durch das Schlüsselwort -inf und inf.

```
>integrate(dnorm, -1.96, 1.96)
>integrate(dnorm, -Inf, Inf)
## a slowly-convergent integral
>f<- function(x) {1/((x+1)*sqrt(x))}
>integrate(f, lower = 0, upper = Inf)
## don't do this if you really want the integral from 0 to
   Inf
integrate(integrand, lower = 0, upper = 10)
integrate(integrand, lower = 0, upper = 100000)
integrate(integrand, lower = 0, upper = 1000000, stop.on.
   error = FALSE)
## fails on many systems
integrate(dnorm, 0, 20000)
integrate(dnorm, 0, Inf)   ## works
```

[math.ethz.ch]

3.4.3 Vor-/Nachteile

Integrate bietet eine bequeme Lösung zur Berechnung von Integralen, dies ist jedoch auf eindimensionale Integrale begrenzt. Desweiteren ist die Fehlerquote bei sehr großen Intervallgrenzen höher aufgrund des Annäherungsverfahrens. [math.ethz.ch] schlägt bei solchen Fällen vor unbegrenzte Integrale zu bilden.

4 CAS Schnittstellen

Ein sehr mächtiges Werkzeug aus dem CRAN-Archiv bilden die Schnittstellen zu vollständigen Compupteralgebrasystemen. Mit diesen sind wir in der Lage nun R zumindest indirekt als ein CAS zu behandeln. Im Folgenden gehen wir grob auf Umgansweise zweier CAS-Interfaces, da dies sonst thematisch den Rahmen sprengen würde. Wer sich weiter informieren will, den weise ich auf meine Quellen hin.

4.1 rYacas

4.1.1 Vorstellung

Yacas steht für Yet Another Computer Algebra System, und ein CAS welches unter der Open-Source GNU-Lizenz veröffentlicht wurde. Der enorme Vorteil bei Yacas besteht darin, dass es seine eigen definierte Programmiersprache besitzt. [Yacas]
So sind wir in der Lage über das R-Paket rYacas aus dem CRAN, Yacas Befehle direkt in der R-Umgebung zu formulieren, die dann an Yacas übergeben werden.

4.1.2 Beispiele

In diesem Beispiel besteht die erste Zeile aus einem mehrfach geschachteltem Aufruf. Wir erstellen zunächst einen factor in R von dem wir mehrere Werte erwarten können. Diesen übergeben wir als R-expression an die Yacas-Call Funktion. In der Yacas-Umgebung ist dies nun eine gültige Funktion. Vorsicht! R erkennt diesen Ausdruck jedoch nur als eine weitere expression! Nun nutzen wir Eval mit der Yacas-Funktion und einem gegebenen Wert dazu und den Funktionswert zu bilden.

```
> f = yacas(expression(Factor(x^3+x^2+x+1))) #Yacas-
    Funktionsaufruf.
> f
expression((x^2 + 1) * (x + 1))
> Eval(f,list(x=1))
[1] 4
> Eval(f,list(x=0))
[1] 1
```

Eine weitere Möglichkeit Funktionen zu definieren ist mittels Sym. Hier legen wir ein global abhängiges Symbol für unsere Funktionen fest. Bilden wie gewohnt eine R-funktion und können diese auslesen. Desweiteren können wir damit einen Yacas-Aufruf zum ableiten bilden, indem wir das Sym-Objekt mit als Parameter übergeben. Die Ableiten wir wieder in Form einer expression zurückgeliefert.

```
x = Sym("x")
f = function(x){(x^3+x^2+x+1)}
> f(0)
[1] 1
> f(1)
[1] 4
> f(42)
[1] 75895
> g = yacas(deriv(f(x), x)) #Yacas-Funktionsaufruf.
> g
expression(3 * x^2 + 2 * x + 1)
```

Integrale bildet man über die integrate ähnlich die mit dem CRAN-Paket.

$$f(x) = -x^2 + 1$$
$$\int_{-1}^{1} f(x)\,\mathrm{d}x = 1,33\overline{33}$$

```
> f = function(x)-x^2+1
> integrate(f,-1,1)
1.333333 with absolute error < 1.5e-14
```

Dies ist auch mit trignonometrischen Funktionen möglich.

$$g(x) = sin(x)$$
$$\int_{0}^{\pi} g(x)\,\mathrm{d}x = 2$$

```
> g = function(x)sin(x)
> g
function(x)sin(x)
> integrate(g,0,pi)
2 with absolute error < 2.2e-14
```

4.1.3 Pretty Printing

Eine komfortable Fähigkeit von rYacas ist das Pretty Printing. Damit können wir mathematische Ausdrücke leserlich gestalten indem wir sie in ASCII oder LaTeX umwandeln.

```
> h = expression(x + x^2/2 + x^3/6 + 1)
> PrettyForm(h)

     2    3
    x    x
x + -- + -- + 1
    2    6
>TeXForm(h)
expression("$x + \frac{x ^{2}}{2}  + \frac{x ^{3}}{6}  + 1$")
```

4.2 rSympy

4.2.1 Vorstellung

Ein weiter bekannten Computeralgebrasystem ist Sympy. Dies ist der CAS-Bereich der Programmiersprache Python, wodurch Sympy für Leute mit Python-Erfahrung einfach zu bedienen ist. Um Sympy nun in R verwenden zu können, gilt dieselbe Prämisse: Wir müssen Sympy aus R direkt ansprechen können, was durch Python jedoch gegeben ist. Das Paket rSympy aus dem CRAN steht für diesen Zweck zur Verfügung. [Sympy.org]

5 Zusammenfassung

5.1 Inhalt

Das CRAN-Archiv bietet also einige Erweiterungen mit denen R in der Lage ist mit Gleichungen und symbolische Audsdrücken umzugehen und diese zu verarbeiten. Nativ kann R Matrixberechnungen durchführen durch einen eigen definierten Datentyp und kann diese auch als Gleichungssystem interpetieren und mit Solve lösen. Mpoly bietet und mit einem neuen Datentyp Optionen zur symbolischen Manipulation. Mit simplr können wir nun Ausdrücke in eine Standardform umformen. Durch das Paket D können wir über Polynomial-funktionen hinaus ableiten und mit Integrate berechnen wir eindimensionale Integrale.

1. Lösung eines Gleichungssystems bilden. [Solve] ✓
2. Matrix-Operationen [Matrix-Datentyp] ✓
3. Arithmetische Operationen auf Polynomen. [mpoly] ✓
4. Vereinfachen von komplizierten symbolischen Ausdrücken in eine Standardform. [simplr] ✓
5. Wertzuweisung der Variablen und Berechnung. [mpoly] ✓
6. Ableiten von symbolischen Ausdrücken. [D] ✓
7. Bilden von Integralen [Integrate] ✓

Durch diese Pakete können wir R als ein primitives Computeralgebrasystem betrachten da die definierten Operationen von einzelnen Paketen implementiert werden.

5.2 Fazit

Zwar bietet R nun Funktionen für grundlegende symbolische Mathematik, jedoch arbeiten die Pakete nicht miteinander zusammen und sind alle getrennt voneinander implementiert, was einem kontinuierlichen workflow entgegensteht. Desweiteren ist R eine interpretierte Programmiersprache, welche für Statistik und Datenverarbeitung entwickelt wurde und somit schwach bei hoher numerischer Berechnung ist. Diese beiden Aspekte werden jedoch durch die CAS-Schnittstellen von R kompensiert ,denn diese bieten Zugang zu ausgereiften Computeralgebrasystemen. Daher ist von einer direkten Zweckentfremdung abzuraten.

Literatur

[CRAN]
https://cran.r-project.org/

[CRAN:Ryacas]
https://cran.r-project.org/web/packages/Ryacas/index.html
http://www.r-bloggers.com

[Yacas] http://www.yacas.org/

[Sympy.org]
http://www.sympy.org/en/index.html

[Planetmath.org]
http://planetmath.org/

[EconBS:D]
http://www.econometricsbysimulation.com/2012/08/
symbolic-differentiation-in-r.html

[mathe-werkstatt]
http://www.mathe-werkstatt.de/themen/cas.htm

[math.ethz.ch]
https://stat.ethz.ch/R-manual/R-devel/library/stats