Univerzitet u Beogradu – Elektrotehnički fakultet (ETF)
Katedra za signale i sisteme

Tehnike obrade biomedicinskih signala 13M051TOBS

Dr Nadica Miljković, vanredna profesorka
kabinet 68, nadica.miljkovic@etf.bg.ac.rs

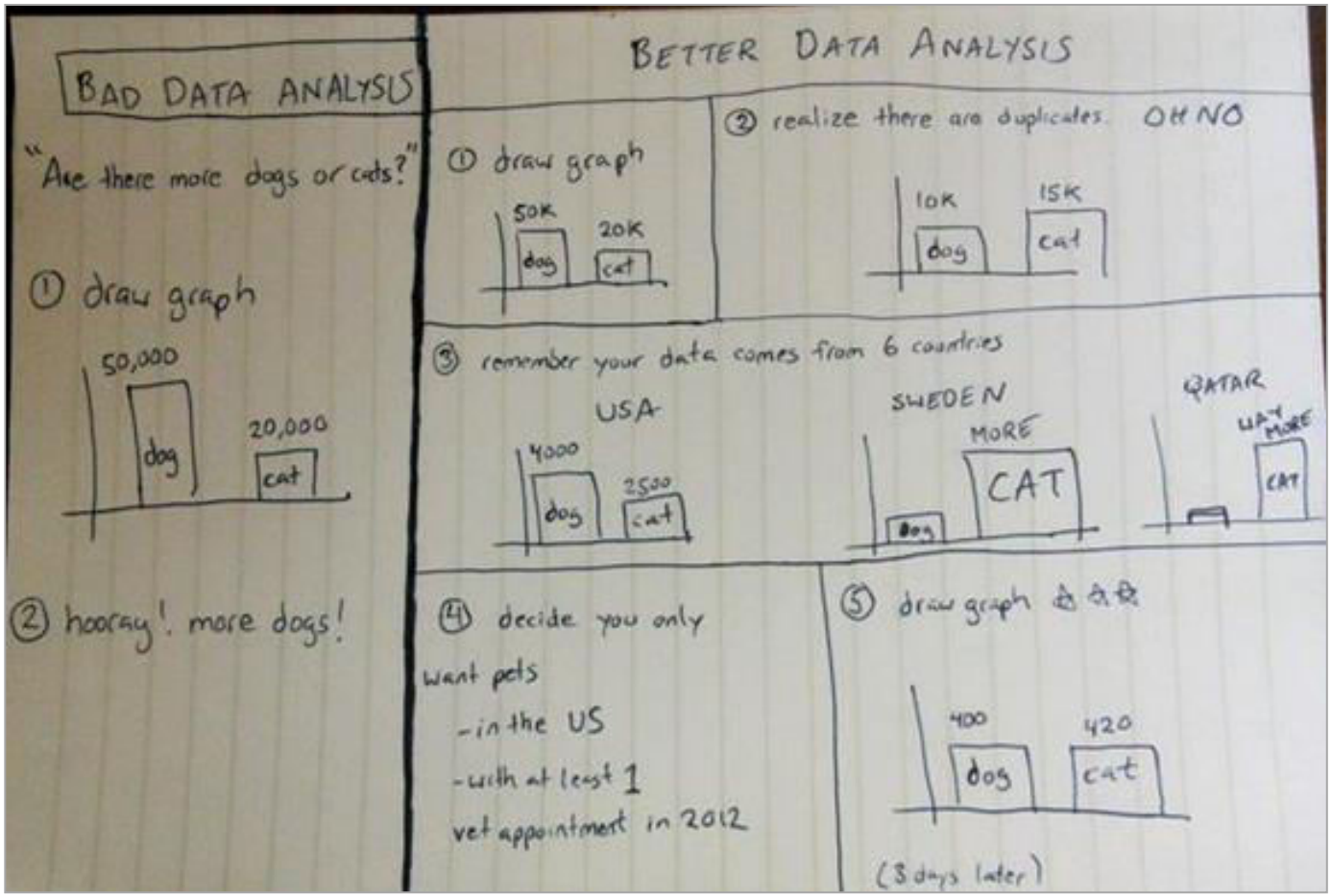
OD PROŠLE SEDMICE - PODSETNIK

Šta će se desiti ako se pozove *rnorm(m = 2, 5)*?

```
> rnorm(m = 2, 5)
[1] 4.083656 2.944312 1.344213 1.493174 2.172898
```

- Rezultat poziva funkcije je prikazan na slici.
- Najpre se traže obavezni argumenti (u ovom slučaju broj 5), a potom se traže argumenti kako je objašnjeno na prethodnom predavanju:
  - proverava da li postoji potpuno podudaranje za neki od argumenata po imenu,
  - proverava da li postoji delimično podudaranje za neki od argumenata po imenu i
  - proverava da li postoji poziciono podudaranje.

Pas i mačka



- Slika sa Twittera: https://pbs.twimg.com/media/CJAJkJyW8AA9gvm.jpg, pristupljeno 30.03.2024.
- Čuvajte se zaključaka. I ono što deluje jednostavno, ne mora da bude.

KONTROLNE FUNKCIJE

Kontrolne funkcije

- Koriste se kao zamena za kontrolne strukture (petlje) u kodu.
- Kontrolne funkcije u R-u su:
  - *lapply()* – koristi se za listu,
  - *sapply()* – isto kao *lapply(),* ali radi redukciju kompleksnosti,
  - *apply()* – koristi se za nizove,
  - *tapply()* – koristi se za podskup vektora i za podatke u formi tabela,
  - *mapply()* – koristi se za matrice i
  - dodatno *split()* – koristi se u kombinaciji sa *lapply()*.

lapply() funkcija

```
> x <- list(prvi = 1:10, drugi = c("l", "i", "s", "t", "a"))
> lapply(x, mean)
$prvi
[1] 5.5

$drugi
[1] NA

Warning message:
In mean.default(X[[i]], ...) :
  argument is not numeric or logical: returning NA
> x1 <- list(prvi = 1:10, drugi = 1:100)
> lapply(x1, mean)
$prvi
[1] 5.5

$drugi
[1] 50.5

> x <- 1:100
> lapply(x, mean)
[[1]]
[1] 1

[[2]]
[1] 2

[[3]]
[1] 3
```

- Dva osnovna argumenta ove funkcije su:
  - lista (ako nije lista, interno se primenjuje *as.list()* funkcija) i
  - funkcija koja se primenjuje na sve elemente liste pojedinačno.
- Izlazni rezultat *lapply()* funkcije je lista.

lapply() funkcija

```
> x3 <- list(prvi = 1:10, drugi = c(NA, 2, 5, 9, 10))
> lapply(x3, mean)
$prvi
[1] 5.5

$drugi
[1] NA

> lapply(x3, mean, na.rm = TRUE)
$prvi
[1] 5.5

$drugi
[1] 6.5
```

- Kako se prosleđuju argumenti funkcije koja se primenjuju na svaki element liste?
- Da li je moguće, bez primene *lapply()* funkcije primeniti *mean()* funkciju na *x3* listu?

lapply() i anonimne funkcije

```
> x4 <- list(prvi = matrix(1:10, 2, 5), drugi = matrix(1:10, 5, 2))
> lapply(x4, function(pom) pom[ ,1])
$prvi
[1] 1 2

$drugi
[1] 1 2 3 4 5

> x4
$prvi
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10

$drugi
     [,1] [,2]
[1,]    1    6
[2,]    2    7
[3,]    3    8
[4,]    4    9
[5,]    5   10

> |
```

- Unutar *lapply()* funkcije argument može da bude i funkcija koju je korisnik definisao.
- Ako je telo funkcije relativno kratko, onda se može definisati u *lapply()* pozivu anonimna funkcija (nema ime).
- Na primeru izdvajanja prve kolone iz elemenata liste je prikazan rad anonimne funkcije.

sapply() funkcija

```
> x4 <- list(prvi = 1, drugi = 1:5, treci = matrix(1:10, 2, 5), cetvrti = matrix(1:10, 5, 2))
> lapply(x4, mean)
$prvi
[1] 1

$drugi
[1] 3

$treci
[1] 5.5

$cetvrti
[1] 5.5

> sapply(x4, mean)
   prvi   drugi   treci cetvrti
    1.0     3.0     5.5     5.5
> |
```

- Ova funkcija radi isto kao *lapply()*, ali je rezultat "jednostavniji".
- Na primeru je prikazana razlika između ove dve funkcije.
- Češće se primenjuje od *lapply()*. Međutim, potrebno je voditi računa o tome da nije uvek pogodno da u kodu postoji funkcija za koju se ne zna koji tip podataka će dati na izlazu. Postoje i druge slične funkcije iz "purrr" paketa.
- Koja je razlika *sapply()* i *lapply()* u primeru sa slike?

sapply() funkcija

```
> class(lapply(x4, mean))
[1] "list"
> class(sapply(x4, mean))
[1] "numeric"
>
```

- U kom smislu jednostavno?
  - Ako je rezultat lista gde je svaki element jedan broj, onda je izlaz niz.
  - Ako je rezultat lista sa elementima istih dužina, onda je rezultat matrica.
  - Ako ne postoji način da se zaključi kako da se pojednostavi rezultat, onda je rezultat *sapply()* funkcije lista kao sa *lapply()*.

split() funkcija

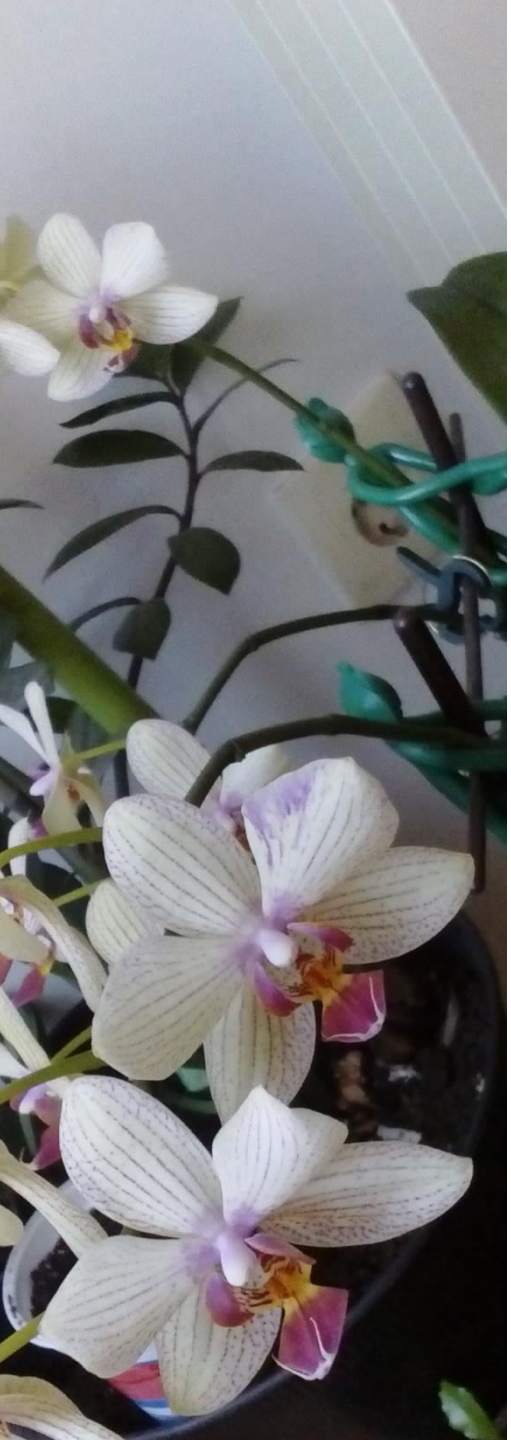
```
> x <- c(1:5, 11:15, 21:25)
> x
 [1]  1  2  3  4  5 11 12 13 14 15 21 22 23 24 25
> f <- gl(3, 5)
> f
 [1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3
Levels: 1 2 3
> sapply(split(x, f), mean)
 1  2  3
 3 13 23
> lapply(split(x, f), mean)
$`1`
[1] 3

$`2`
[1] 13

$`3`
[1] 23

>
```

- *split()* funkcija "razdvaja" objekte (od npr. liste) na podskupove prema definisanom kriterijumu (npr. kategorička promenljiva). Argument *drop* (može imati TRUE ili FALSE vrednost) i daje naznaku da li u rezultatu funkcije *split()* treba ili ne treba da postoje prazni nivoi/podskupovi.
- Funkcija *gl()* generiše nivoe faktora (eng. *generate factor levels*).
- Funkcija *split()* se obično koristi u kombinaciji sa *lapply()* odnosno *sapply()* funkcijama.
- Koja je razlika između *lapply()* i *sapply()* funkcija sa slike?

split() i *data.frame* podaci

```
> head(PlantGrowth, 3)
  weight group
1   4.17  ctrl
2   5.58  ctrl
3   5.18  ctrl
> unique(PlantGrowth$group)
[1] ctrl trt1 trt2
Levels: ctrl trt1 trt2
> dat <- split(PlantGrowth$weight, PlantGrowth$group)
> head(dat)
$ctrl
 [1] 4.17 5.58 5.18 6.11 4.50 4.61 5.17 4.53 5.33 5.14

$trt1
 [1] 4.81 4.17 4.41 3.59 5.87 3.83 6.03 4.89 4.32 4.69

$trt2
 [1] 6.31 5.12 5.54 5.50 5.37 5.29 4.92 6.15 5.80 5.26

> class(dat)
[1] "list"
> class(dat$ctrl)
[1] "numeric"
>
> rez <- sapply(dat, mean, na.rm = TRUE)
> rez
 ctrl  trt1  trt2
5.032 4.661 5.526
```

- Najčešće se *split()* koristi na *data frame* podacima.
- U biblioteci *dataset* postoje ugrađeni podaci koji se mogu koristiti.
- Spisak svih dostupnih podataka se može naći na: https://stat.ethz.ch/R-manual/R-devel/library/datasets/html/00Index.html (pristupljeno 30.03.2024). Dodatne informacije o *dataset* paketu se dobijaju komandom *library(help = "datasets")*.
- Na slici je dat primer ovih funkcija na podacima *PlantGrowth*.
- Ne treba učitavati biblioteku za korišćenje ovih podataka.



Podaci *PlantGrowth*

- Ovi podaci su ugrađeni u R-u (https://cran.r-project.org/package=datasets.load, pristupljeno 30.03.2024).
- Preuzeti su iz knjige:
  - Dobson, Annette J. *Introduction to statistical modeling*. Springer, 2013.
- Oznaka *ctrl* označava kontrolnu grupu biljaka, a oznake *trt1* i *trt2* označavaju tretman 1 i tretman 2.
- Težina biljaka se odnosi na suvu materiju.

split() u više nivoa

```
> faktor1 <- gl(2, 3)
> faktor1
[1] 1 1 1 2 2 2
Levels: 1 2
> faktor2 <- gl(3, 2)
> faktor2
[1] 1 1 2 2 3 3
Levels: 1 2 3
>
> interaction(faktor1, faktor2)
[1] 1.1 1.1 1.2 2.2 2.3 2.3
Levels: 1.1 2.1 1.2 2.2 1.3 2.3
> list(faktor1, faktor2)
[[1]]
[1] 1 1 1 2 2 2
Levels: 1 2

[[2]]
[1] 1 1 2 2 3 3
Levels: 1 2 3

>
```

- U slučaju da postoje podskupovi podskupova, onda je zgodno koristiti funkciju *interaction()*.
- Umesto *interaction()* funkcije može se koristiti *list()*.
- Sa ovim funkcijama se definiše broj grupa za koje je potrebno uraditi razdvajanje podataka u podgrupe.
- Relativno jednostavan primer je prikazan na slici. U ovom slučaju je pogodno koristiti *drop* argument funkcije *split()*. Ovaj argument označava da se izostavljaju prazni nivoi.

tapply() funkcija

```
> x <- c(1:5, 11:15, 21:25)
> x
 [1]  1  2  3  4  5 11 12 13 14 15 21 22 23 24 25
> f <- gl(3, 5)
> f
 [1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3
Levels: 1 2 3
> tapply(x, f, mean)
 1  2  3
 3 13 23
> sapply(split(x, f), mean)
 1  2  3
 3 13 23
> tapply(x, f, mean, simplify = FALSE)
$`1`
[1] 3

$`2`
[1] 13

$`3`
[1] 23

> lapply(split(x, f), mean)
$`1`
[1] 3

$`2`
[1] 13

$`3`
[1] 23

>
```

- Ova funkcija omogućava da se u petlji / matrično primeni neka metoda/funkcija na podskupu ulaznih podataka.
- Da, radi slično kao kombinacija *s/lapply()* i *split()* funkcija.
- Postoji i argument *simplify* koji odgovara *sapply()* funkciji.
- Koja je podrazumevana vrednost ovog argumenta?

apply() funkcija

```
> matrica <- matrix(1:10, 2, 5)
> matrica
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
> apply(matrica, 2, mean)
[1] 1.5 3.5 5.5 7.5 9.5
> apply(matrica, 1, mean)
[1] 5 6
> rowMeans(matrica)
[1] 5 6
> colMeans(matrica)
[1] 1.5 3.5 5.5 7.5 9.5
>
```

- Služi da se primeni funkcija (najčešće anonimna) na niz podataka (vrste/kolone matrice). Na slici je dat primer za računanje srednje vrednosti svake kolone matrice.
- Ako se koristi *apply()* funkcija sa *mean()* i/ili *sum()* funkcijom, onda postoje i ugrađene funkcije: *rowSums()*, *rowMeans()*, *colSums()* i *colMeans()* (ove funkcije su daleko brže od *apply()* funkcije, a mogu računati i medijanu).

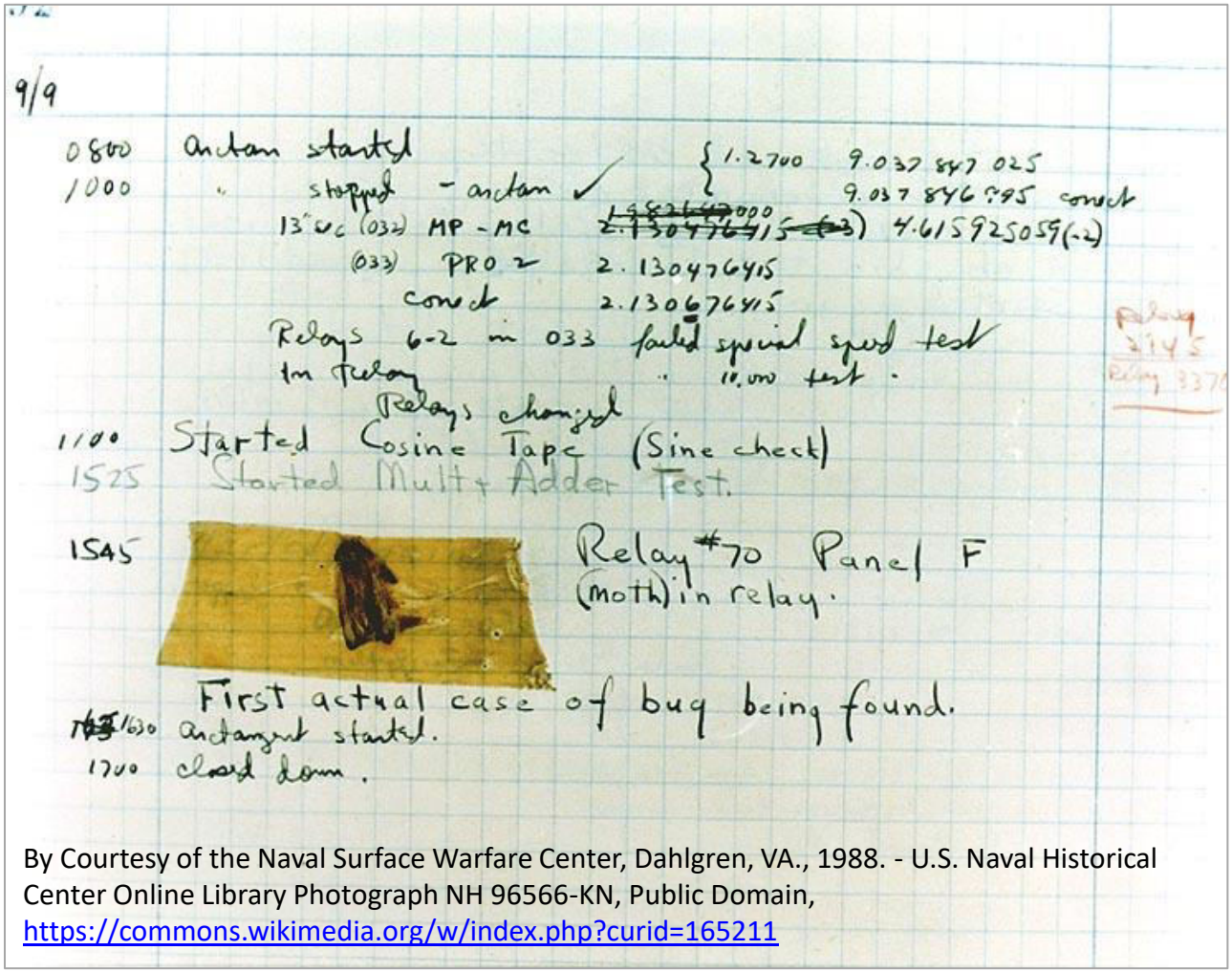
mapply() funkcija

```
> # mapply funkcija
> ?rep
> rep(2)
[1] 2
> rep(2, 4)
[1] 2 2 2 2
> rep(c(3, 5, 4), 12)
 [1] 3 5 4 3 5 4 3 5 4 3 5 4 3 5 4 3 5 4 3 5 4 3 5 4 3 5 4 3 5 4 3 5 4 3 5 4
> rep(1:3, 3)
[1] 1 2 3 1 2 3 1 2 3
> rep(3, 1:3)
Error in rep(3, 1:3) : invalid 'times' argument
> mapply(rep, 3, 1:3)
[[1]]
[1] 3

[[2]]
[1] 3 3

[[3]]
[1] 3 3 3
```

- Funkcija *rep()* omogućava da se ponavljaju prosleđeni argumenti željeni broj puta. Podrazumevano se ponavlja argument jednom.
- Vrlo često je potrebno primeniti ovu funkciju kada se želi niz/matrica ili neki drugi element sa prethodno definisanim vrednostima.
- Međutim, nije moguća vektorizacija broja ponavljanja.
- Primenom *mapply()* funkcije ovaj problem je rešen. Pogledati kod.

DEBAGOVANJE

Debagovanje



By Courtesy of the Naval Surface Warfare Center, Dahlgren, VA., 1988. - U.S. Naval Historical Center Online Library Photograph NH 96566-KN, Public Domain, https://commons.wikimedia.org/w/index.php?curid=165211

- Debagovanje predstavlja metodu pronalaska i rešavanja situacija koje nastaju kao rezultat projektovanog koda i sprečavaju željeni rad tog koda (https://en.wikipedia.org/wiki/Debugging).
- Početak korišćenja ovog termina se vezuje za Harvard Univerzitet i Mark II računar iako postoje dokazi da je termin korišćen još u 19. veku. Pogledati sliku.

Debagovanje

```
> log(-1)
[1] NaN
Warning message:
In log(-1) : NaNs produced
```

- Kada se program ne izvršava na odgovarajući način, ispisuje se:
  - poruka (eng. *message*) – ne dolazi do prekidanja izvršavanja koda, ali dolazi do obaveštenja koje je rezultat *message()* funkcije,
  - upozorenje (eng. *warning*) – slično kao na slici, samo je obaveštenje rezultat *warning()* funkcije,
  - greška (eng. *error*) – obaveštenje da je došlo do problema, program se zaustavlja i obaveštenje i zaustavljanje su rezultat *stop()* funkcije i
  - uslov (eng. *condition*) – korisnik može sam da definiše pod kojim uslovima dolazi do obaveštenja odnosno zaustavljanja programa.

Primer greške

```
> # primer greške
> parNepar <- function(x) {
+         if (x %% 2 == 0) {
+                 print("Paran broj.")
+         } else {
+                 print("Neparan broj.")
+         }
+         invisible(x)
+ }
> ?invisible
> parNepar(13)
[1] "Neparan broj."
> parNepar(10)
[1] "Paran broj."
> parNepar(NA)
Error in if (x%%2 == 0) { : missing value where TRUE/FALSE needed
>
```

- Na slici je dat primer funkcije i ulazni argument koji rezultuje greškom.
- Čemu služi funkcija *invisible(x)*? Služi da izlaz funkcije, ako se dodeli nekoj promenljivoj bude jednak *x*, inače će biti jednak *string*-u koji se štampa.

Popravka #1

```
> # popravka #1
> parNepar <- function(x) {
+         if (is.na(x)) {
+                 print("Uneli ste nedostajuću vrednost.")
+         } else if (x %% 2 == 0) {
+                 print("Paran broj.")
+         } else {
+                 print("Neparan broj.")
+         }
+         invisible(x)
+ }
> parNepar(25)
[1] "Neparan broj."
> parNepar(2)
[1] "Paran broj."
> parNepar(NA)
[1] "Uneli ste nedostajuću vrednost."
> parNepar(c(1, 6))
[1] "Neparan broj."
Warning messages:
1: In if (is.na(x)) { :
  the condition has length > 1 and only the first element will be used
2: In if (x%%2 == 0) { :
  the condition has length > 1 and only the first element will be used
>
```

- Sa korekcijama kao na slici moguće je pozvati funkciju za argument koji je nedostajuća vrednost (NA).
- Iako funkcija sada ne javlja grešku, treba imati na umu da rezultat funkcije može da se prosleđuje drugom delu koda te greška može da propagira.

Popravka #2

```
> parNeparV <- Vectorize(parNepar)
> parNeparV(c(1, 6))
[1] "Neparan broj."
[1] "Paran broj."
[1] 1 6
> parNeparV
function (x)
{
    args <- lapply(as.list(match.call())[-1L], eval, parent.frame())
    names <- if (is.null(names(args)))
        character(length(args))
    else names(args)
    dovec <- names %in% vectorize.args
    do.call("mapply", c(FUN = FUN, args[dovec], MoreArgs = list(args[!dovec]),
        SIMPLIFY = SIMPLIFY, USE.NAMES = USE.NAMES))
}
<environment: 0x00000000164d7920>
>
```

- Vektorizacija funkcije *parNepar()* primenom *Vectorize()* funkcije.
- Primetiti da je sada telo funkcije izmenjeno.

Kako izbeći greške?

- Preporuke:
  - Proveriti ulazni argument funkcije?
  - Proveriti kako je funkcija “pozvana”?
  - Uporediti očekivan sa dobijenim izlazom funkcije.
  - Da li je očekivan = realan rezultat?
  - Da li je moguće ponoviti* rezultat funkcije (i očekivan i dobijen)?
- Ovo ne radi! 
- Ovo je očekivano, a ovo je rezultat! 

*Primer: korišćenjem *set.seed()* funkcije.

Alati za debagovanje

- “Bezgrešan” kod ne postoji...
- Ugrađeni alati za debagovanje u R-u su:
  - *traceback()*: daje korisniku informaciju o “call stack”-u, odnosno o informaciji do “korene” funkcije,
  - *debug()*: funkcija je u *debug mode*-u, može da se po jedna linija izvršava jedna za drugom,
  - *browser()*: slično kao *debug()*, samo što omogućava upravljanje linijama koda,
  - *trace()*: da se *debug* kod doda u funkciju bez menjanja koda (posebno ako se kod nalazi u nekom paketu, pa mu nije zgodno pristupiti) i
  - *recover()*: modifikuje poruku greške (eng. *error behavior*).
- Jednostavniji način da se upravlja greškama je da se omogući prikaz nekih promenljivih pomoću eksplicitnog štampanja (nije idealno).
- Mnogi smatraju da alati za debagovanje nisu najpoželjnija opcija.

Alati za debagovanje: *traceback()*

```
> print(y)
Error in print(y) : object 'y' not found
> traceback()
1: print(y)
>
>
> sd(y)
Error in is.data.frame(x) : object 'y' not found
> traceback()
3: is.data.frame(x)
2: var(if (is.vector(x) || is.factor(x)) x else as.double(x), na.rm = na.rm)
1: sd(y)
>
>
> print("Ovde nema greške.")
[1] "Ovde nema greške."
> traceback()
3: is.data.frame(x)
2: var(if (is.vector(x) || is.factor(x)) x else as.double(x), na.rm = na.rm)
1: sd(y)
> y <- 2
> print(y)
[1] 2
> traceback()
3: is.data.frame(x)
2: var(if (is.vector(x) || is.factor(x)) x else as.double(x), na.rm = na.rm)
1: sd(y)
>
```

- Ako postoji greška u nekoj funkciji koja se poziva onda je *traceback()* veoma koristan – njega prvo treba pogledati.
- Mana *traceback()* funkcije: mora da se pozove odmah nakon koda u kome je došlo do greške.
- Šta se desi ako se pozove nakon koda koji ne javlja grešku? Štampa poslednju grešku.

Alati za debagovanje: *debug()*

- Funkcija *debug()* se može primeniti na bilo koju funkciju.
- Na kraju poziva ove funkcije se otvara *browser prompt*. Kucanjem “n” (i ENTER! sa tastature) izvršava se jedna linija koda i ulazi u novu liniju*.
- Iz *Debug mod*-a se izlazi ili kada se naiđe na grešku ili pozivom funkcije *undebug()*.
- *Debug mod* se može pokretati unutar drugog *Debug mode*-a: posebno je zgodno za ugnježdene funkcije.

*c – nastavlja sa izvršavanjem funkcije i zaustavlja se sa pojavom greške, Q – izlazi se iz *Browser*-a.

Alati za debagovanje: *recover*

```
> options(error = recover)
> sd(x)
Error in is.data.frame(x) : object 'x' not found

Enter a frame number, or 0 to exit

1: sd(x)
2: var(if (is.vector(x) || is.factor(x)) x else as.double(x), na.rm = na.rm)
3: is.data.frame(x)

Selection:
```

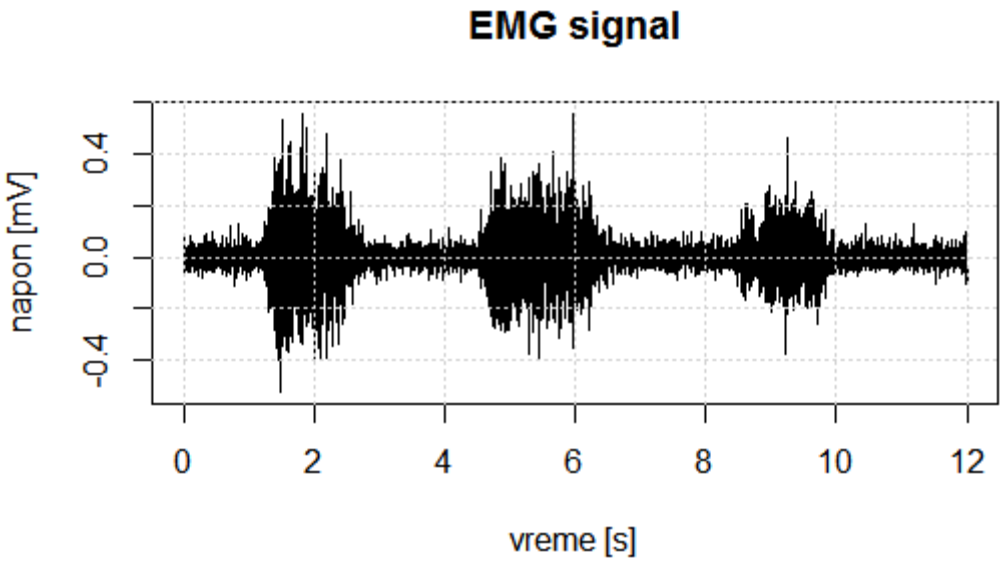
- Ova opcija se koristi za modifikaciju prikaza i ponašanja koda prilikom pojave greške.
- Korisno je da se zaustavi na mestu pojave greške, kako bi se dalje istražila ta lokacija i greška uklonila.
- Primer poziva *recover* opcije je prikazan na slici.

PRETPROCESIRANJE SIGNALA

EMG (elektromiografija)

- EMG predstavlja elektrofiziološki signal sa mišića, odnosno električnu aktivnost mišića. Za njegovo merenje potrebno je primeniti pojačanje (uobičajeno se koristi instrumentacioni pojačavač sa pojačanjem od 100 do 5000 puta).
- Amplituda ovog signala je par mV, a frekvencijski sadržaj je od 0 do maksimalno 500 Hz, ako se meri površinski. Na slici je prikazano kako se Ag/AgCl pretvarači postavljaju na mišić *brachioradialis* i *extensor digitorum* leve ruke za merenje EMG signala.
- Signali koji se koriste u ovoj studiji mereni su na Elektrotehničkom fakultetu u januaru 2017. godine. Sniman je signal sa *brachioradialis* mišića za tri pokreta fleksije (savijanja) šake, kao na slici. Pojačanje signala je bilo 1000 puta.

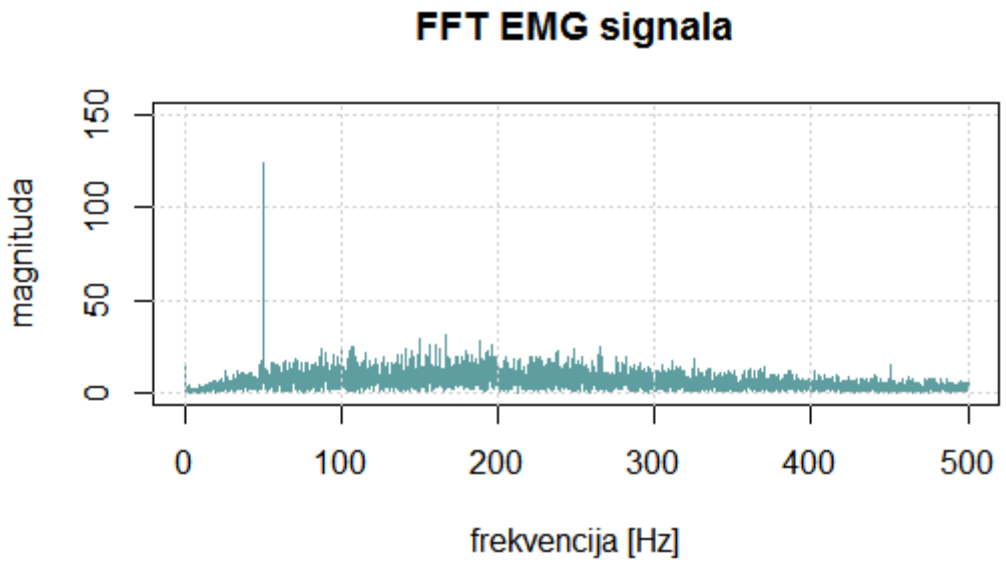
Signal u vremenskom domenu




```
> fs <- 1000 #u Hz
> dat <- read.csv("EMG.csv")
> sig1 <- dat$EMG
> time <- seq(0, (length(sig1)/fs - 1/fs), 1/fs)
> plot(time, sig1, type = "l", xlab = "vreme [s]", ylab = "napon [mV]",
+      main = "EMG signal")
> grid()
> head(sig1)
[1]  0.020  0.017  0.043  0.017 -0.024 -0.026
>
```

- Ovaj signal u vremenskom domenu “podseća” na audio signal.
- Kolika je frekvencija odabiranja? Da li je to dovoljno velika frekvencija odabiranja?
- Koliko je mišićnih kontrakcija prikazano na slici?

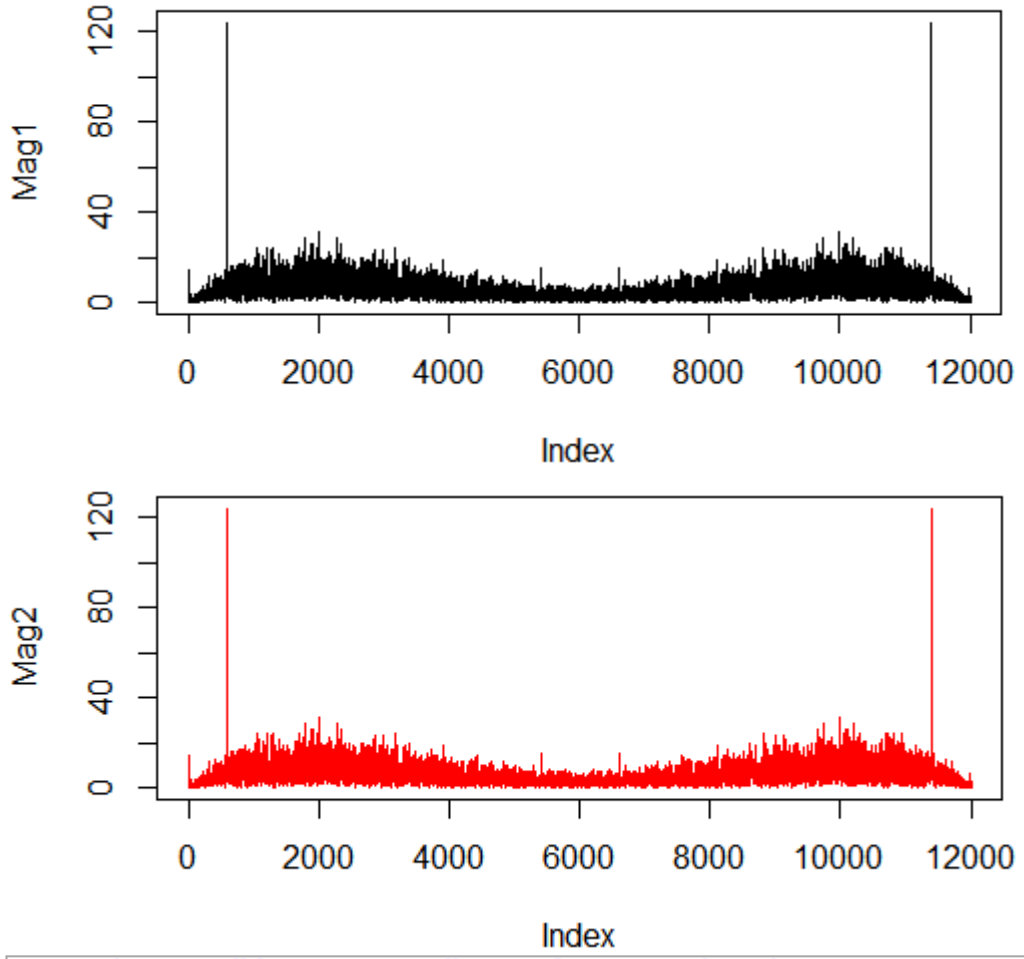
Signal u frekvencijskom domenu



- Na slici je prikazana Furijeova transformacija signala.
- Da li je u spektru signala vidljiv neki šum?

```
> sig1FFT <- fft(sig1)
> class(sig1FFT)
[1] "complex"
> head(sig1FFT)
[1] 14.5890000+0.0000000i  0.7409584+0.8645697i  0.0464307-0.4066154i  4.1558902-4.2782738i
[5] -2.0494716+0.9968703i -1.5166775+4.0740635i
> sig1Mag <- Mod(sig1FFT)
> length(sig1Mag)
[1] 12000
> sig1Mag <- sig1Mag[1:(length(sig1Mag)/2)] #magn_1
> length(sig1Mag)
[1] 6000
> fosa <- 1:length(sig1Mag)/max(time)
> plot(fosa, sig1Mag, type = "l",
+      ylim = c(0, 150), xlim = c(0, 500),
+      xlab = "frekvencija [Hz]", ylab = "magnituda",
+      main = "FFT EMG signala", col = "cadetblue")
> grid(col = "lightgrey")
>
```

Dva načina za računanje magnitude



- Na slici su prikazane obe magnitude i kod za njihovo generisanje i računanje.
- Da li je prikazana jednostrana ili dvostrana FFT?
- Da li su ova dva signala ista?

```
> # dva načina za računanje magnitude
> Mag1 <- Mod(sig1FFT)
> Mag2 <- sqrt(Re(sig1FFT)*Re(sig1FFT)+Im(sig1FFT)*Im(sig1FFT))
>
> plot(Mag1, type = "l")
> plot(Mag2, col = "red", type = "l")
```

Poređenje dve magnitude u R-u

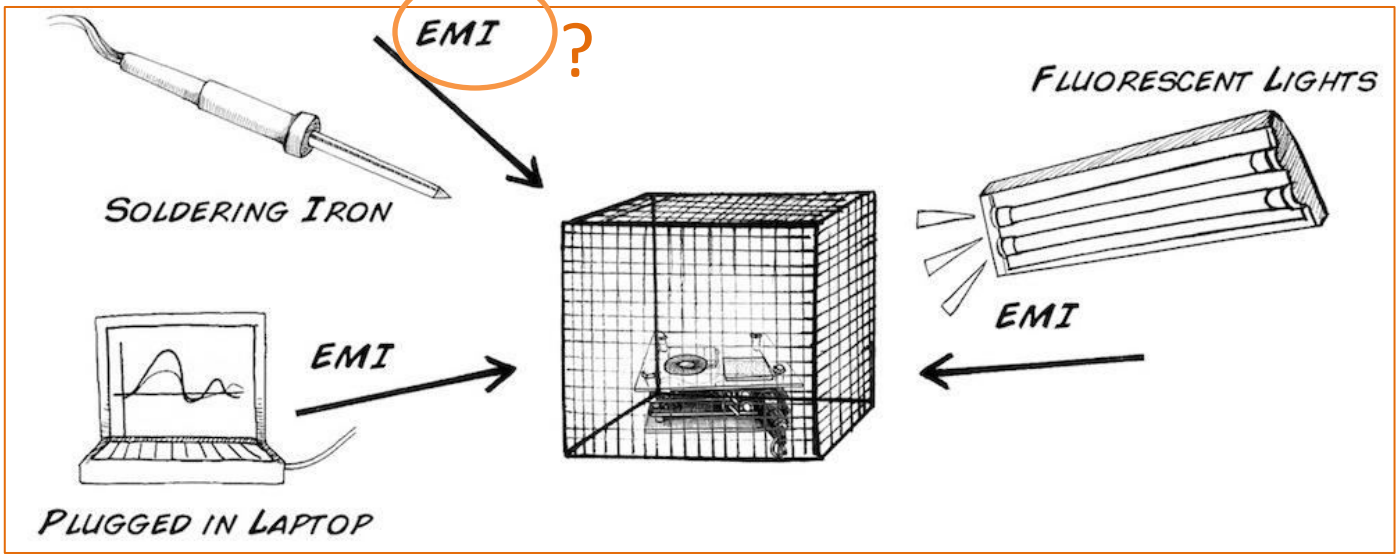
```
> identical(Mag1, Mag2)
[1] FALSE
> Indeksi <- which(Mag1 != Mag2)
> Indeksi
[1]   671  3171 10680
> Mag1[Indeksi] - Mag2[Indeksi]
[1]  8.881784e-16 -4.440892e-16  8.881784e-16
> sum(Mag1[-Indeksi] - Mag2[-Indeksi])
[1] 0
>
```

- Zašto nastaje razlika između ove dve magnitude?
- Da li će računanje na jedan ili na drugi način, imati uticaja na poređenje dva signala?

Obrada biomedicinskih signala?

- Po čemu je specifična obrada biomedicinskih signala u odnosu na druge signale?
  - kompleksnost sistema na kome se meri i
  - pretežno neinvazivna merenja (podložna šumu/artifaktima).
- Najveći problem za pretprocesiranje biomedicinskih signala predstavlja šum napajanja 50/60 Hz!
- Kako odabrati metodu analize signala?
  - Pre primene odgovarajuće tehnike za obradu signala, važno je znati opšte karakteristike signala (što je više moguće).
  - Cilj analize može usmeriti izbor metode za analizu signala.
  - Iskustvo.

Faradejev kavez i šum napajanja



https://backyardbrains.com/experiments/faraday#prettyPhoto/0/ (pristupljeno 30.03.2024), *Fair Use*.

- Bolje sprečiti nego lečiti.
- Retko se koristi, ali se može koristiti.
- Smanjuje udobnost prilikom merenja.
- Kompleksnost primene.
- Uglavnom se koristi za signale manje od µV.
- Više na https://en.wikipedia.org/wiki/Faraday_cage.

Filtriranje signala

```
> # paket signal
> library(signal)
> f1 <- butter(3, c(49/(fs/2), 51/(fs/2)), type = "stop")
> sig1filt <- filtfilt(f1, sig1)
>
```

- Funkcije za analizu signala (eng. *signal processing functions*) po ugledu na kod u Matlabu i GNU Octaveu se nalaze u paketu “signal”.
- Detalje o ovom paketu je moguće pogledati na CRAN-u: https://cran.r-project.org/package=signal (pristupljeno 30.03.2024).
- Filtriranje signala se vrši kao na slici.
- Zašto je korišćen *Butterworth* filtar?

Računanje koeficijenata filtra

$$Y(z) = \frac{b(1) + b(2)z^{-1} + \ldots + b(nb+1)z^{-nb}}{1 + a(2)z^{-1} + \ldots + a(na+1)z^{-na}} X(z)$$

```
> f1
$b
[1]   0.9875122  -5.6351906  13.6815175 -18.0667531  13.6815175
[6]  -5.6351906   0.9875122

$a
[1]   1.0000000  -5.6825488  13.7387486 -18.0664565  13.6241305
[6]  -5.5881290   0.9751803

attr(,"class")
[1] "Arma"
> class(f1)
[1] "Arma"
> f1$b
[1]   0.9875122  -5.6351906  13.6815175 -18.0667531  13.6815175
[6]  -5.6351906   0.9875122
> f1$a
[1]   1.0000000  -5.6825488  13.7387486 -18.0664565  13.6241305
[6]  -5.5881290   0.9751803
>
```

- *b* i *a* koeficijenti predstavljaju koeficijente za imenilac i brojilac za prikaz funkcije prenosa filtra u Z domenu.
- Kako izgleda funkcija prenosa ovog *notch* filtra?

Kako odabrati tip filtra?

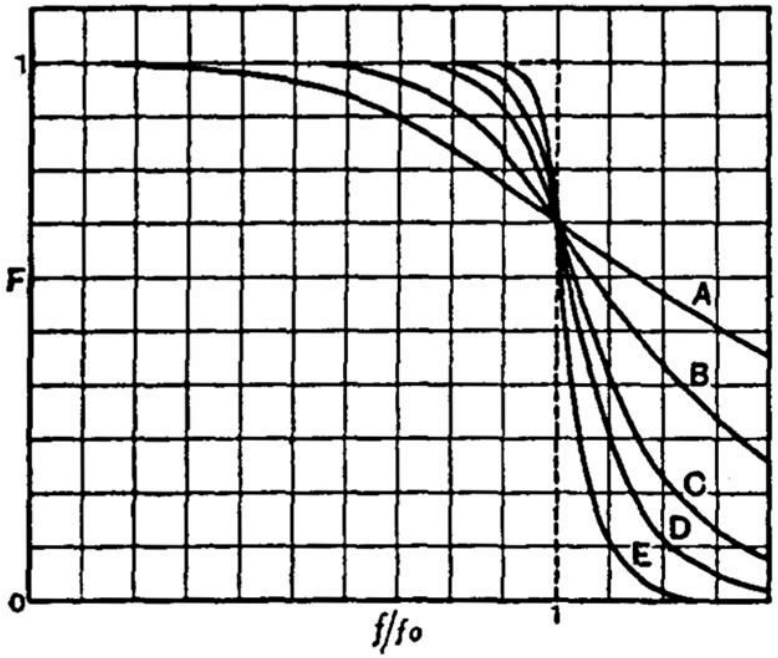
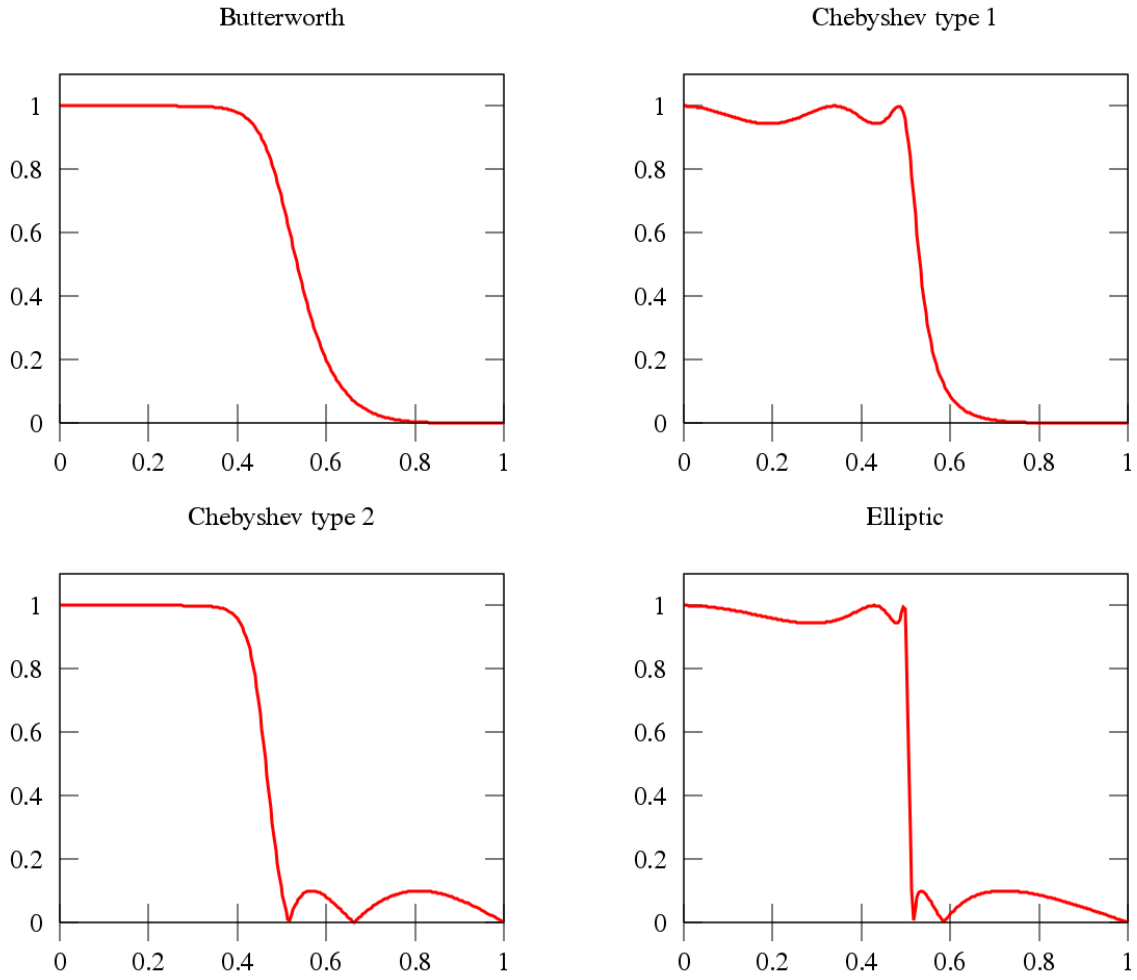


Fig. 3.

By Stephen Butterworth 1885-1958 - Experimental Wireless page 536-541, Public Domain, https://en.wikipedia.org/w/index.php?curid=26098721

- Ne postoji uputstvo.
- Za razne primene se koriste razni filtri.
- *Butterworth* filter se najčešće koriste za biomedicinske signale.
- Više na https://en.wikipedia.org/wiki/Butterworth_filter.

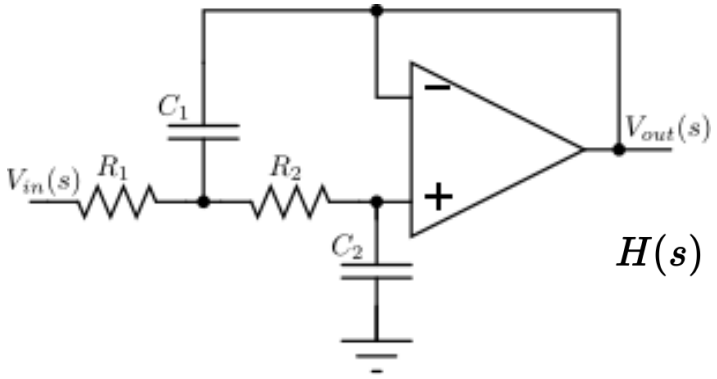
Zašto *Butterworth*?



Slika: By Alessio Damato - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=427738

- Na slici su prikazani filtri 5. reda.
- Koja je razlika između prikazanih filtara?

Kako se realizuje ovaj filter?



Slika: By Guillaume Simard - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=4275147

$$H(s) = \frac{V_{out}(s)}{V_{in}(s)} = \frac{1}{1 + C_2(R_1 + R_2)s + C_1 C_2 R_1 R_2 s^2}.$$

butter {signal} R Documentation

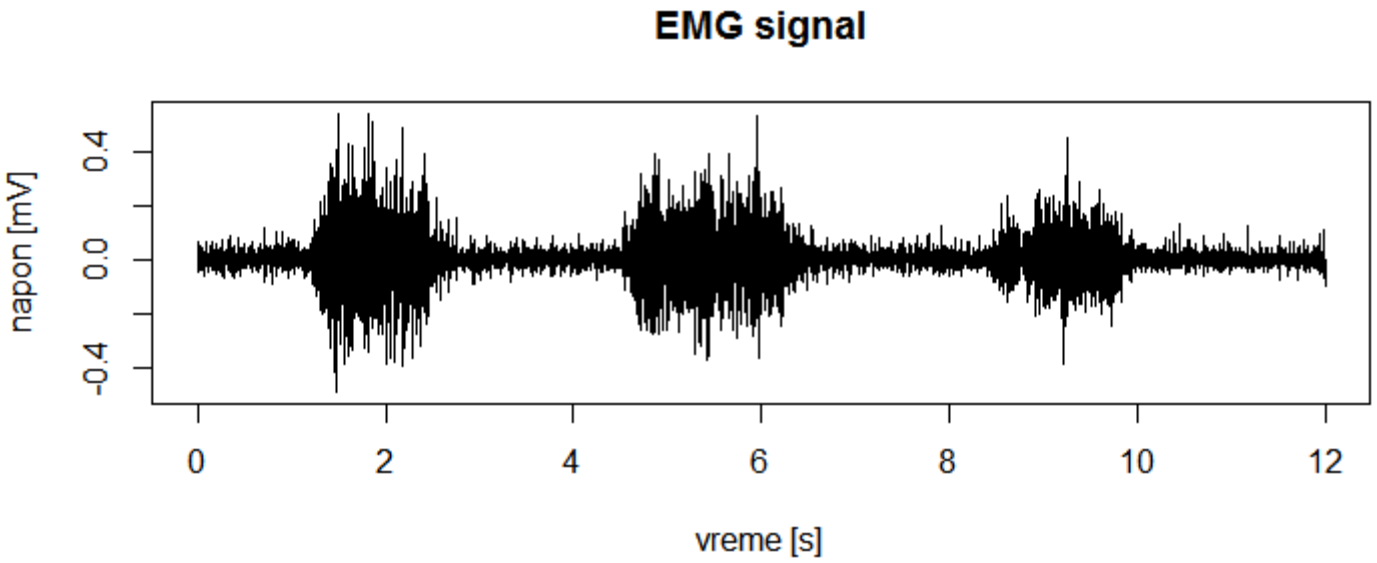
Generate a Butterworth filter.

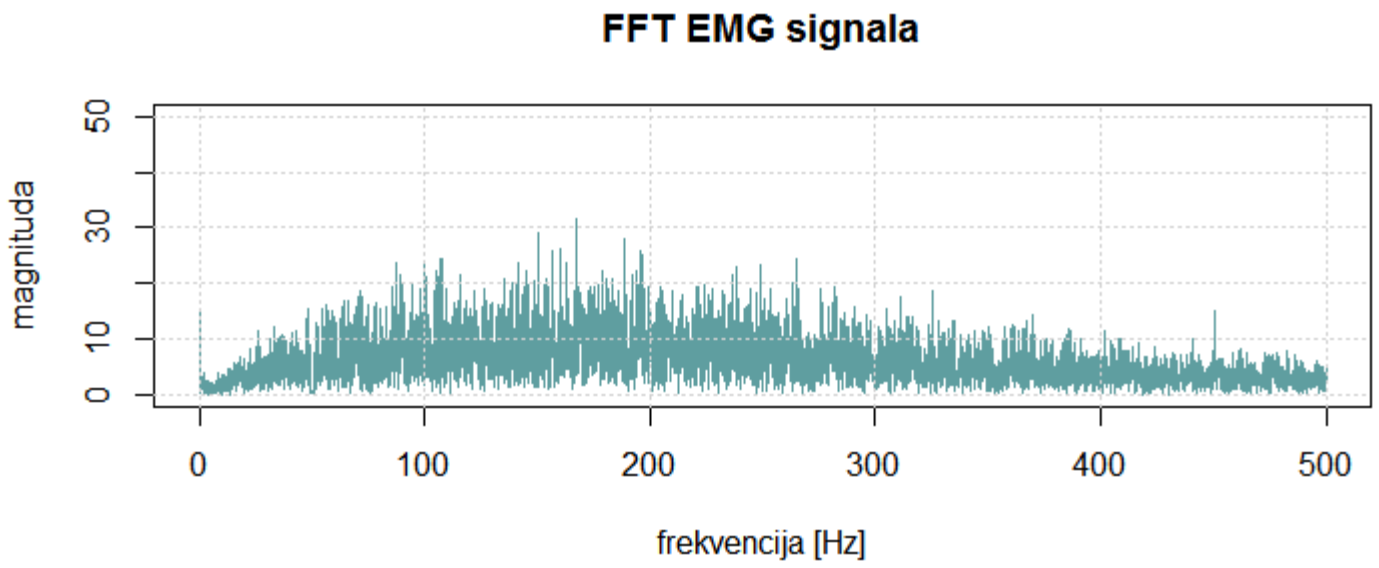
Description

Generate Butterworth filter polynomial coefficients.

- Na slici je prikazana hardverska realizacija *Butterworth* filtra preko *Sallen-Key* topologije koja koristi aktivne i pasivne električne komponente za realizaciju linearnih analognih filtara. Na slici je prikazan filter 2. reda.
- Odabirom R i C komponenti podešava se frekvencija odsecanja filtra.
- U R-u se koristi *butter()* funkciju u kombinaciji sa *filtfilt()* funkcijom. Na slici je dato uputstvo.

Kako izgleda filtriran signal?





Da li je šum otklonjen?

```
> plot(time, sig1filt, type = "l", xlab = "vreme [s]",
ylab = "napon [mV]",
+      main = "EMG signal")
> sig1filtFFT <- Mod(fft(sig1filt))
> sig1filtFFT <- sig1filtFFT[1:(length(sig1filtFFT)/2)]
> plot(fosa, sig1filtFFT, type = "l",
+      ylim = c(0, 50), xlim = c(0, 500),
+      xlab = "frekvencija [Hz]", ylab = "magnituda",
+      main = "FFT EMG signala", col = "cadetblue")
> grid(col = "lightgrey")
>
```

- Kod za prikaz signala sa prethodnog slajda je dat na slici.
- Naravno, nema šuma napajanja na 50 Hz.
- Na kojoj frekvenciji se nalazi maksimum ovog signala?
- Da li su korišćeni neki analogni filtri prilikom merenja ovog signala?

filtfilt() funkcija

```
filtfilt <- function(filt, ...) UseMethod("filtfilt")

filtfilt.default <- function(filt, a, x, ...)  {
    y = filter(filt, a, c(x, numeric(2 * max(length(a), length(filt)))))
    y = rev(filter(filt, a, rev(y)))[seq_along(x)]
    y
}

filtfilt.Arma <- function(filt, x, ...) # IIR
  filtfilt(filt$b, filt$a, x)

filtfilt.Ma <- function(filt, x, ...) # FIR
  filtfilt(as.Arma(filt), x)

filtfilt.Zpg <- function(filt, x, ...) # Zero-pole-gain ARMA representation
  filtfilt(as.Arma(filt), x)
```

R-Forge@R-project.org

Powered by ViewVC 1.0.0

- Koja je razlika između *filter()* i *filtfilt()* metode?
- Na slici je prikazan originalan slobodan kod za *filtfilt()* funkciju u R-u, https://r-forge.r-project.org/scm/viewvc.php/pkg/R/filtfilt.R?view=markup&root=signal, pristupljeno 30.03.2024.

PROFILISANJE R KODA

Profilisanje R koda

- Alati za profilisanje omogućavaju da se na sistematičan način prouči koliko je vreme izvršavanja različitih delova koda (https://en.wikipedia.org/wiki/Profiling_(computer_programming).
- Najčešće se koristi za optimizaciju koda.
- VAŽNO: Kada je kod napisan (čitko!), testiran i debagovan, tek onda se pristupa profilisanju.
- Optimizacija se nikada ne zasniva na pretpostavkama, već na objektivnim činjenicama.
- Profilisanje može značajno da poveća efikasnost koda.
- U R-u postoje i osnovne funkcije kao što je *Rprof()*, ali i posebni paketi koji se mogu koristiti za profilisanje.

system.time()/system.time({})

```
> system.time(readLines("http://automatika.etf.rs/index.php/sr/"))
   user  system elapsed
   0.42    0.08    0.73
Warning message:
In readLines("http://automatika.etf.rs/index.php/sr/") :
  incomplete final line found on 'http://automatika.etf.rs/index.php/sr/'
>
```

- Argument ove funkcije je kod čije vreme izvršavanja (u s) bi trebalo izmeriti.
- Ako postoji greška u kodu (!?), onda je rezultat vreme koje prođe do pojave greške.
- Rezultat je R objekat klase *proc_time*.
- Postoje tri "vrste" proteklog vremena:
  - *system time*: vreme koje je potrebno procesoru (misli se na izvršavanje), zavisi od operativnog sistema (OS)
  - *user time* UT: vreme koje je potrebno procesoru (misli se na instrukcije), zavisi od OS
  - *elapsed time* ET (eng. *wall clock time*): realno vreme
- UT ~ ET za jednostavne zadatke/komande
- ET > UT kada CPU ima čekanje
- ET < UT kada postoji *multi-core* procesor (procesor sa više jezgara)
- MANA: *system.time()* se ne može koristiti da odredi koji deo koda se duže izvršava, on jednostavno meri vreme na delu koda koji se odabere.

Rprof() funkcija

```
> Rprof()
> sig1filt <- filtfilt(f1, sig1)
> Rprof(NULL)
>
```

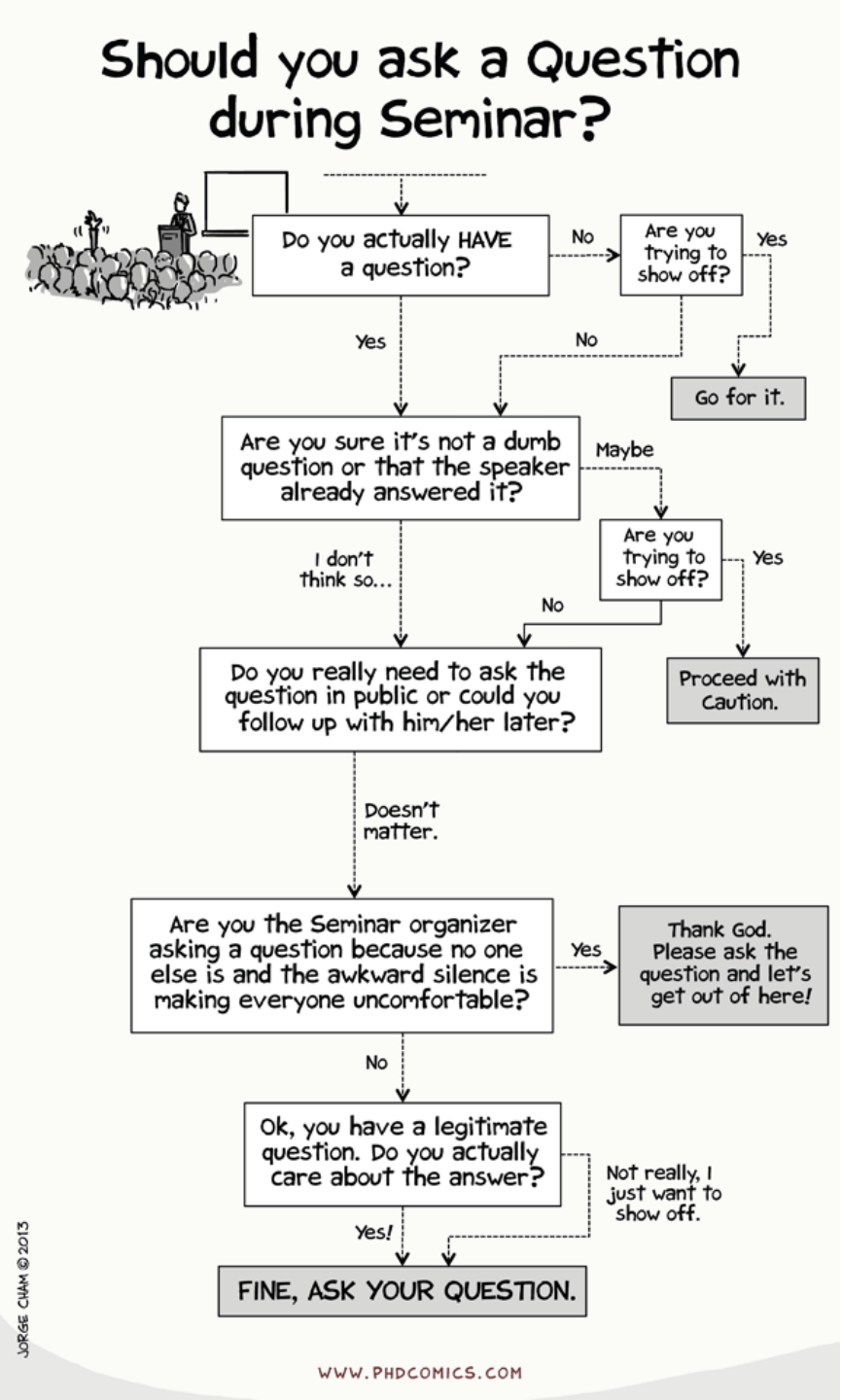
- Ova funkcija pokreće proceduru profilisanja koda.
- Ne sme se koristiti u kombinaciji sa *system.time()* funkcijom.
- Dodatno: koristi se *summaryRprof()* funkcija koja sumira rezultat *Rprof()*.
- Procedura se pokreće sa *Rprof()* i zaustavlja se *Rprof(NULL)*.
- Podrazumevano vreme odabiranja je 0.02 s (na svakih 0.02 s se ispisuje deo koda) -> ako je program relativno brz, *Rprof()* nije koristan, ali tada nije ni potreban.
- Kao rezultat *Rprof()* pokretanja u radnom direktorijumu se snima datoteka *Rprof.out*. Nakon toga, da bi se prikazao ovaj rezultat koristi se *summaryRprof()* funkcija.

summaryRprof() funkcija

```
> Rprof()
> sig1filt <- filtfilt(f1, sig1)
> Rprof(NULL)
> summaryRprof()
$by.self
    self.time self.pct total.time total.pct
"c"      0.02      100       0.02       100

$by.total
                  total.time total.pct self.time self.pct
"c"                     0.02       100      0.02      100
"filter"                0.02       100      0.00        0
"filter.default"        0.02       100      0.00        0
"filtfilt"              0.02       100      0.00        0
"filtfilt.Arma"         0.02       100      0.00        0
"filtfilt.default"      0.02       100      0.00        0
"rev"                   0.02       100      0.00        0
"stats::filter"         0.02       100      0.00        0

$sample.interval
[1] 0.02

$sampling.time
[1] 0.02

>
```

- Rezultat pokretanja ove funkcije dat je na slici.
- Postoje dve metode za normalizaciju podataka:
  - *by.total*: deli se vreme provedeno u svakoj funkciji sa totalnim/ukupnim vremenom i
  - *by.self*: slično kao gore, ali se najpre oduzme vreme provedeno u funkcijama u *call stack*-u.

PAR KORISNIH SUGESTIJA

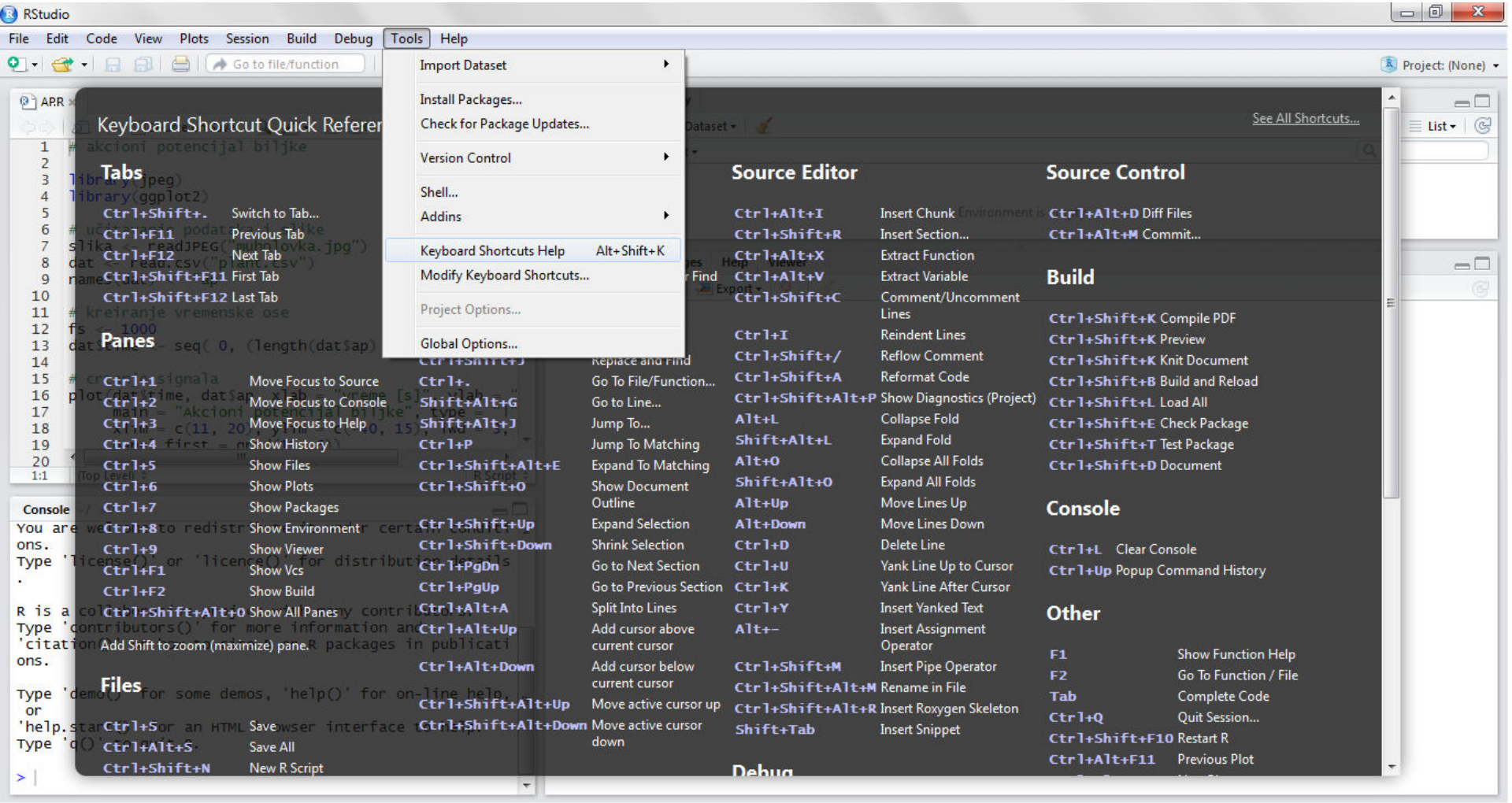


Online help

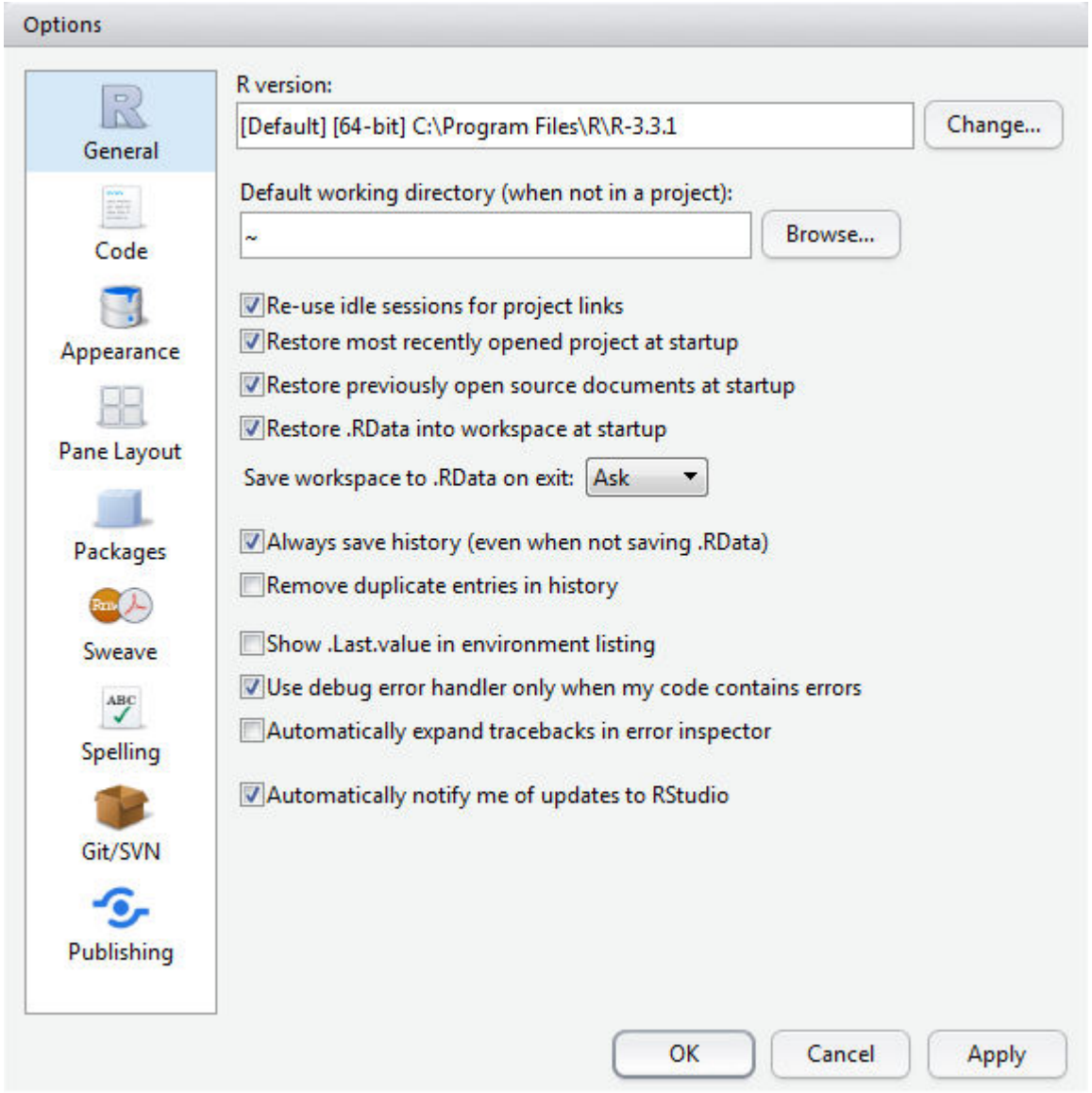
- Pre pitanja:
  - Analiza problema (≠ izazova)
  - Google, https://www.google.rs/
- Kako postaviti pitanje na pametan način?
  - Eric S. Raymond's Home Page: http://www.catb.org/~esr/faqs/smart-questions.html, pristupljeno 30.03.2024.
- Poštujte bonton i pravila!
- *Google alati:*
  - Za naučnu literaturu, https://scholar.google.com/
  - Patente, https://patents.google.com/
  - Knjige, https://books.google.com/
  - Podatke, https://datasetsearch.research.google.com/
  - Trendove (šta drugi pretražuju), https://trends.google.com/trends/
  - Google Books Ngram Viewer, https://books.google.com/ngrams
  - Ima i Google Lens i drugih alata

Slika: http://phdcomics.com/comics/archive_print.php?comicid=1632 (pristupljeno 2017. godine), *Fair Use*.

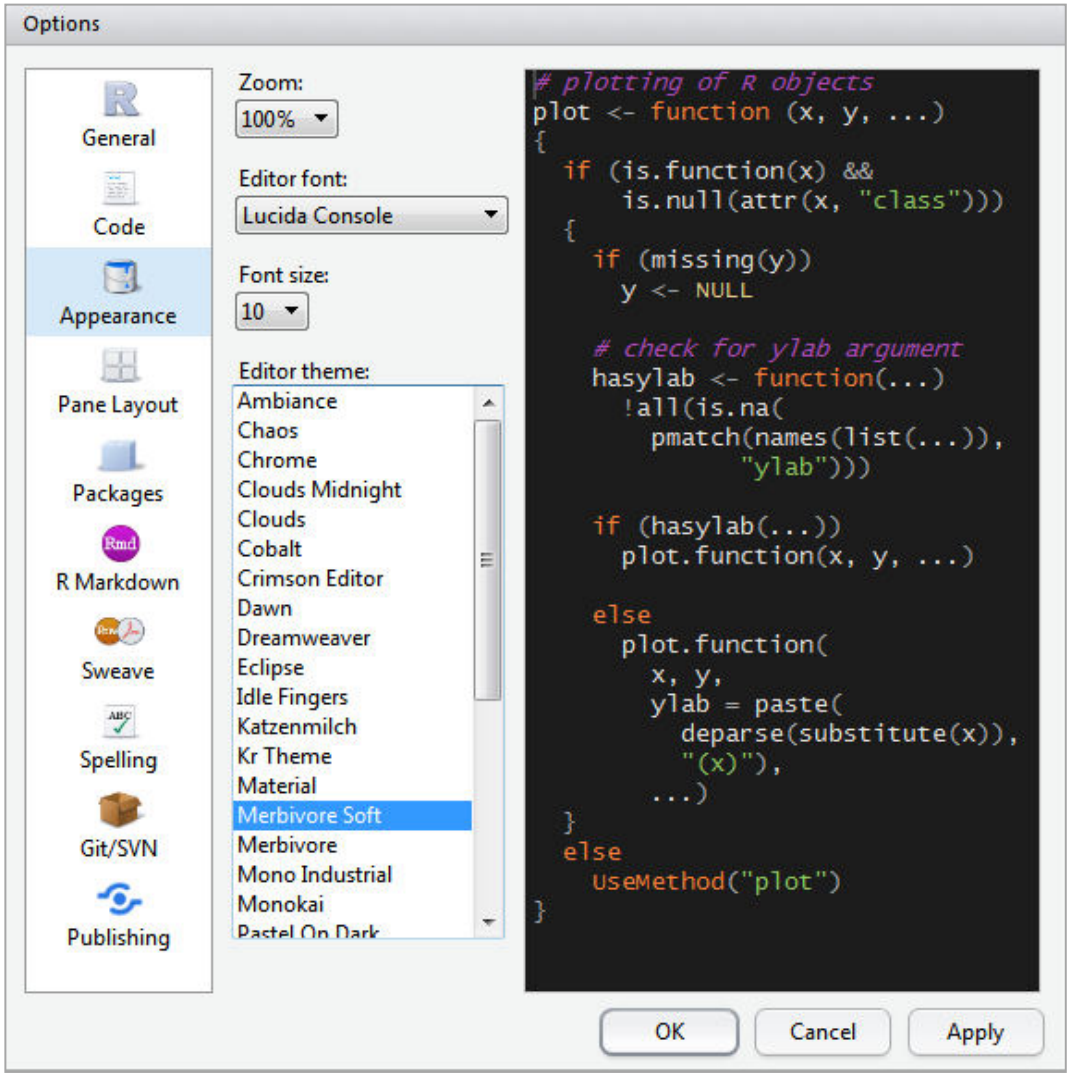
Korisno?


RStudio
File Edit Code View Plots Session Build Debug Tools Help
Import Dataset
Install Packages...
Check for Package Updates...
Version Control
Shell...
Addins
Keyboard Shortcuts Help Alt+Shift+K
Modify Keyboard Shortcuts...
Project Options...
Global Options...
Keyboard Shortcut Quick Reference
See All Shortcuts...
Tabs
Ctrl+Shift+. Switch to Tab...
Ctrl+F11 Previous Tab
Ctrl+F12 Next Tab
Ctrl+Shift+F11 First Tab
Ctrl+Shift+F12 Last Tab
Panes
Ctrl+1 Move Focus to Source
Ctrl+2 Move Focus to Console
Ctrl+3 Move Focus to Help
Ctrl+4 Show History
Ctrl+5 Show Files
Ctrl+6 Show Plots
Ctrl+7 Show Packages
Ctrl+8 Show Environment
Ctrl+9 Show Viewer
Ctrl+F1 Show Vcs
Ctrl+F2 Show Build
Ctrl+Shift+Alt+0 Show All Panes
Add Shift to zoom (maximize) pane.
Files
Ctrl+S Save
Ctrl+Alt+S Save All
Ctrl+Shift+N New R Script
Source Editor
Ctrl+Alt+I Insert Chunk
Ctrl+Shift+R Insert Section...
Ctrl+Alt+X Extract Function
Ctrl+Alt+V Extract Variable
Ctrl+Shift+C Comment/Uncomment Lines
Ctrl+I Reindent Lines
Ctrl+Shift+/ Reflow Comment
Ctrl+Shift+A Reformat Code
Ctrl+Shift+Alt+P Show Diagnostics (Project)
Alt+L Collapse Fold
Shift+Alt+L Expand Fold
Alt+O Collapse All Folds
Shift+Alt+O Expand All Folds
Alt+Up Move Lines Up
Alt+Down Move Lines Down
Ctrl+D Delete Line
Ctrl+U Yank Line Up to Cursor
Ctrl+K Yank Line After Cursor
Ctrl+Y Insert Yanked Text
Alt+- Insert Assignment Operator
Ctrl+Shift+M Insert Pipe Operator
Ctrl+Shift+Alt+M Rename in File
Ctrl+Shift+Alt+R Insert Roxygen Skeleton
Shift+Tab Insert Snippet
Ctrl+. Go To File/Function...
Ctrl+Shift+G Go to Line...
Ctrl+Shift+J Jump To...
Ctrl+P Jump To Matching
Ctrl+Shift+Alt+E Expand To Matching
Ctrl+Shift+O Show Document Outline
Ctrl+Shift+Up Expand Selection
Ctrl+Shift+Down Shrink Selection
Ctrl+PgDn Go to Next Section
Ctrl+PgUp Go to Previous Section
Ctrl+Alt+A Split Into Lines
Ctrl+Alt+Up Add cursor above current cursor
Ctrl+Alt+Down Add cursor below current cursor
Ctrl+Shift+Alt+Up Move active cursor up
Ctrl+Shift+Alt+Down Move active cursor down
Source Control
Ctrl+Alt+D Diff Files
Ctrl+Alt+M Commit...
Build
Ctrl+Shift+K Compile PDF
Ctrl+Shift+K Preview
Ctrl+Shift+K Knit Document
Ctrl+Shift+B Build and Reload
Ctrl+Shift+L Load All
Ctrl+Shift+E Check Package
Ctrl+Shift+T Test Package
Ctrl+Shift+D Document
Console
Ctrl+L Clear Console
Ctrl+Up Popup Command History
Other
F1 Show Function Help
F2 Go To Function / File
Tab Complete Code
Ctrl+Q Quit Session...
Ctrl+Shift+F10 Restart R
Ctrl+Alt+F11 Previous Plot
Debug

Tools/Global Options



Još opcija?


Options
General
Code
Appearance
Pane Layout
Packages
R Markdown
Sweave
Spelling
Git/SVN
Publishing
Zoom:
100%
Editor font:
Lucida Console
Font size:
10
Editor theme:
Ambiance
Chaos
Chrome
Clouds Midnight
Clouds
Cobalt
Crimson Editor
Dawn
Dreamweaver
Eclipse
Idle Fingers
Katzenmilch
Kr Theme
Material
Merbivore Soft
Merbivore
Mono Industrial
Monokai
Pastel On Dark
# plotting of R objects
plot <- function (x, y, ...)
{
if (is.function(x) &&
is.null(attr(x, "class")))
{
if (missing(y))
y <- NULL
# check for ylab argument
hasylab <- function(...)
!all(is.na(
pmatch(names(list(...)),
"ylab")))
if (hasylab(...))
plot.function(x, y, ...)
else
plot.function(
x, y,
ylab = paste(
deparse(substitute(x)),
"(x)"),
...)
}
else
UseMethod("plot")
}
OK
Cancel
Apply

REZIME

Rezime

Take-home messages

- Kontrolne funkcije su korisne zbog svoje kompaktne forme.
  - Kao argumente moguće je koristiti anonimne funkcije.
  - Vrlo često se koriste u kombinaciji sa *split()* funkcijom, ali i drugim funkcijama.
- Postoje tri glavne indikacije da je došlo do greške ili neočekivanog događaja: poruka, upozorenje i greška. Samo greška zaustavlja izvršavanje koda.
  - **Alati za debagovanje mogu biti od koristi, ali nisu zamena za kritičko razmišljanje.**
- Pretprocesiranje je važno za otklanjanje šuma, primenjuje se i kod EMG signala.
- Zgodno je raditi profilisanje na pojedinačnim funkcijama.
  - Funkcije za profilisanje rade samo sa R kodom, ali ne sa C-om i Fortran-om.
- Literatura korišćena za programiranje u R-u: Peng, R. D. (2016). *R programming for data science* (pp. 86-181). Leanpub.

Za kraj

Slika: Arcade Column od gaspi *yg; Flickr https://www.flickr.com/photos/gaspi/34968120/; CC BY-BN-ND 2.0

“Premature optimization is the root of all evil!”,
prof. Donald Ervin Knuth, https://en.wikiquote.org/wiki/Donald_Knuth.