Navodila za pripravo domačih nalog

Ta dokument vsebuje navodila za pripravo domačih nalog. Navodila so napisana za programski jezik Julia. Če uporabljate drug programski jezik, navodila smiselno prilagodite.

Kontrolni seznam

Spodaj je seznam delov, ki naj jih vsebuje domača naloga.

- koda (`src\DomacaXY.jl`)
- testi (`test\runtests.jl`)
- dokument `README.md`
- demo skripta, s katero ustvarite rezultate za poročilo
- poročilo v formatu PDF

Preden oddate domačo nalogo, uporabite naslednji *kontrolni seznam*:

- vse funkcije imajo dokumentacijo
- testi pokrivajo večino kode
- *README* vsebuje naslednje:
  - ime in priimek avtorja
  - opis naloge
  - navodila kako uporabiti kodo
  - navodila, kako pognati teste
  - navodila, kako ustvariti poročilo
- *README* ni predolg
- poročilo vsebuje naslednje:
  - ime in priimek avtorja
  - splošen(matematičen) opis naloge
  - splošen opis rešitve
  - primer uporabe (slikice prosim :-)

Kako pisati in kako ne

V nadaljevanju je nekaj primerov dobre prakse, kako pisati kodo, teste in poročilo. Pri pisanju besedil je vedno treba imeti v mislih, komu je poročilo namenjeno.

Pisec naj uporabi empatijo do bralca in naj poskuša napisati zgodbo, ki ji bralec lahko sledi. Tudi, če je pisanje namenjeno strokovnjakom, je dobro, če je čim več besedila razumljivega tudi širši publiki. Tudi strokovnjaki radi beremo besedila, ki jih hitro razumemo. Zato je dobro začeti z okvirnim opisom z malo formulami in splošnimi izrazi. V nadaljevanju lahko besedilo stopnjujemo k vedno večjim podrobnostim.

Določene podrobnosti, ki so povezane s konkretno implementacijo, brez škode izpustimo.

Opis rešitve naj bo okviren

Opis rešitve naj bo zgolj okviren. Izogibajte se uporabi programerskih izrazov ampak raje uporabljajte matematične. Na primer izraz uporabimo for zanko, lahko nadomestimo s postopek ponavljamo. Od bralca zahteva splošen opis manj napora in dobi širšo sliko. Če želite dodati izpeljave, jih napišite z matematičnimi formulami, ne v programskem jeziku. Koda sodi zgolj v del, kjer je opisana uporaba za konkreten primer.

Dobro! *Splošen opis algoritma*

Algoritem za LU razcep smo prilagodili tridiagonalni strukturi matrike. Namesto trojne zanke smo uporabili le enojno, saj je pod pivotnim elementom neničelen le en element. Časovna zahtevnost algoritma je tako z $\mathcal{O}(n^3)$ padla na zgolj $\mathcal{O}(n)$.

Slabo! *Podrobna razlaga kode, vrstico po vrstico*

V programu za LU razcep smo uporabili for zanko od 2 do velikosti matrike. V prvi vrstici zanke smo izračunali `L.s[i]`, tako da smo element `T.s[i]` delili z `U.z[i-1]`. Nato smo izračunali diagonalni element, tako da smo uporabili formulo `U.d[i]-L.s[i]*U.d[i-1]`. Na koncu zanke smo vrnili matriki `L` in `U`.

Podrobnosti implementacije ne sodijo v poročilo

Podrobnosti implementacije so razvidne iz kode, zato jih nima smisla ponavljati v poročilu. Algoritme opišete okvirno, tako da izpustite podrobnosti, ki niso nujno potrebne za razumevanje. Podrobnosti lahko dodate, v nadaljevanju, če mislite, da so nujne za razumevanje.

Dobro! *Algoritem opišemo okvirno, podrobnosti razložimo kasneje*

V matriki želimo eleminirati spodnji trikotnik. To dosežemo tako, da stolpce enega za drugim preslikamo s Hausholderjevimi zrcaljenji. Za vsak stolpec poiščemo vektor, preko katerega bomo zrcalili. Vektor poiščemo tako, da bo imela zrcalna slika ničle pod diagonalnim elementom.

Tu lahko z razlago zaključimo. Če želimo dodati podrobnosti, pa jih navedemo za okvirno idejo.

Dobro! *Podrobnosti sledijo za okvirno razlago*

Vektor zrcaljenja dobimo kot

$$u = \left[s(k) + A_{k,k}, A_{k+1,k}, ...A_{n,k}\right],$$

kjer je $s(k) = \text{sign}\left(A_{k,k}\right) * \|A(k:n,k)\|$. Podmatriko $A(k:n, k+1:n)$ prezrcalimo preko vektorja u, tako da podmatriki odštejemo matriko

$$2u\frac{u^T A(k:n,k+1:n)}{u^T u}.$$

Na k-tem koraku prezrcalimo le podmatriko $k:n \times k:n$, ostali deli matrike pa ostanejo nespremenjeni.

Takojšnje razlaganje podrobnosti, brez predhodnega opisa osnovne ideje, ni dobro. Bralec težko loči, kaj je zares pomembno in kaj je zgolj manj pomembna podrobnost.

Slabo! *Takoj dodamo vse podrobnosti, ne da bi razložili zakaj*

Za vsak k, poiščemo vektor $u = \left[s(k) + A_{k,k}, A_{k+1,k}, ...A_{n,k}\right]$, kjer je $s(k) = \text{sign}\left(A_{k,k}\right) * \|\left[A_{k,k}, ..., A_{n,k}\right]\|$.

Nato matriko popravimo

$$A(k:n,k+1:n) = A(k:n,k+1:n) - 2 * u * \frac{u^T * A(k:n,k+1:n)}{u^T * u}.$$

Če implementacija vsebuje posebnosti, kot na primer uporaba posebne podatkovne strukture ali algoritma, jih lahko opišemo v poročilu. Vendar pazimo, da bralca ne obremenjujemo s podrobnostmi.

Dobro! *Posebnosti implementacije opišemo v grobem in se ne spuščamo v podrobnosti*
Za tridiagonalne matrike definiramo posebno podatkovno strukturo `Tridiag`, ki hrani le neničelne elemente matrike. Julia omogoča, da LU razcep tridiagonalne matrike, implementiramo kot specializirano metodo funkcije `lu` iz paketa `LinearAlgebra`. Pri tem upoštevamo posebnosti tridiagonalne matrike in algoritem za LU razcep prilagodimo tako, da se časovna in prostorska zahtevnost zmanjšata na $\mathcal{O}(n)$.

Pazimo, da v poročilu ne povzemamo direktno posameznih korakov kode.

Slabo! *Opisovanje, kaj počnejo posamezni koraki kode, ne sodi v poročilo.*
Za tridiagonalne matrike definiramo podatkovni tip `Tridiag`, ki ima 3 atribute `s`, `d` in `z`. Atribut `s` vsebuje elemente pod diagonalo, ... LU razcep implementiramo kot metodo za funkcijo `LinearAlgebra.lu`. V `for` zanki izračunamo naslednje: 1. element `l[i]=a[i, i-1]/a[i-1, i-1]` 2. ...

Kako pisati teste

Nekaj nasvetov, kako lahko testiramo kodo.

- Na roke izračunajte rešitev za preprost primer in jo primerjajte z rezultati funkcije.
- Ustvarite testne podatke, za katere je znana rešitev. Na primer za testiranje kode, ki reši sistem `Ax=b`, izberete `A` in `x` in izračunate desne strani `b=A*x`.
- Preverite lastnost rešitve. Za enačbe `f(x)=0`, lahko rešitev, ki jo izračuna program preprosto vstavite nazaj v enačbo in preverite, če je enačba izpolnjena.
- Red metode lahko preverite tako, da naredite simulacijo in primerjate red

metode z redom programa, ki ga eksperimentalno določite.

- Če je le mogoče, v testih ne uporabljamo rezultatov, ki jih proizvede koda sama. Ko je koda dovolj časa v uporabi, lahko rezultate kode same uporabimo za regresijske teste.

Pokritost kode s testi

Pri pisanju testov je pomembno, da testi izvedejo vse veje v kodi. Delež kode, ki se izvede med testi, imenujemo pokritost kode (angl. Code Coverage). V juliji lahko pokritost kode dobimo, če dodamo argument `coverage=true` metodi `Pkg.test`:

```
julia> import Pkg; Pkg.test("DomacaXY"; coverage=true)
```

Zgornji ukaz bo za vsako datoteko iz mape `src` ustvaril ustrezno datoteko s končnico `.cov`, v kateri je shranjena informacija o tem, kateri deli kode so bili uporabljeni med izvajanjem testov.

Za poročanje o pokritosti kode lahko uporabite paket Coverage.jl. Povzetek o pokritosti kode s testi lahko pripravite z naslednjim programom:

```
using Coverage
cov = process_folder("DomacaXY")
pokrite_vrstice, vse_vrstice = get_summary(cov)
delez = pokrite_vrstice / vse_vrstice
println("Pokritost kode s testi: $(round(delez*100))%.")
```

Priprava zahteve za združitev na Github

Za lažjo komunikacijo predlagam, da rešitev domače naloge postavite v svojo vejo in ustvarite zahtevo za združitev (*Pull request* na Githubu oziroma *Merge request* na Gitlabu). V nadaljevanju bomo opisali, kako to storiti, če repozitorij z domačimi nalogami gostite na Githubu. Postopek za Gitlab in druge platforme je podoben.

Preden začnete z delom, ustvarite vejo na svoji delovni kopiji repozitorija in jo potisnete na Github ali Gitlab. Ime veje naj bo `domača-X`, se pravi `domaca-1` za 1. domačo nalogo in tako naprej. To storite z ukazom

```
$ git checkout -b domaca-1
$ git push -u origin domaca-1
```

Stikalo `-u` pove git-u, da naj z domačo vejo sledi veji na Githubu/Gitlabu.

Med delom sproti dodajate vnose z `git commit` in jih prenesete na splet z ukazom `git push`. Ko je domača naloga končana, na Githubu ustvarite zahtevo za združitev (angl. Pull request).

- Kliknete na zavihek `Pull requests` in nato na zelen gumb `Create pull request`.
- Na desni strani izberete vejo `domaca-1` in kliknete na gumb `Create draft pull request`.
- Ko je koda pripravljena na pregled, kliknite na gumb `Ready for review`.
- V komentarju za novo ustvarjeno zahtevo povabite asistenta k pregledu. To storite tako, da v komentar dodate uporabniško ime asistenta (npr. `@mojZlobniAsistent`).

```
@mojZlobniAsistent Prosim za pregled.
```

Pri domačih nalogah se posvetujte s kolegi

Nič ni narobe, če za pomoč pri domači nalogi prosite kolega. Seveda morate kodo in poročilo napisati samo, lahko pa kolega prosite za pregled ali za pomoč, če vam kaj ne dela.

Domačo nalogo tudi napišete v skupini, vendar morate v tem primeru rešiti toliko različnih nalog, kot je študentov v skupini.