Zadaća 5

Ova zadaća nosi **8 poena** (plus opcionalna mogućnost da se dobije i još jedan dodatni poen). Rok za predaju je **nedjelja**, **28. I 2024.** do kraja dana. Zadaće se predaju putem Zamgera. Broj poena koji student dobija ovisi od broja korektno implementiranih stavki koje se traže u zadaći.

Zadatak 1 (4 poena)

U ovom zadatku, potrebno je da sastavite skupinu funkcija koje će služiti kao podrška numeričkom rješavanju nelinearnih jednačina. Prva među njima je funkcija "`BracketRoot`" koja ima sljedeći prototip:

```
template <typename FunType>
  bool BracketRoot(FunType f, double x0, double &a, double &b, double hinit = 1e-5,
    double hmax = 1e10, double lambda = 1.4);
```

Ova funkcija pokušava da tehnikom "lovljenja" opisanoj na predavanjima ogradi neku nulu funkcije, tj. da pronađe interval $[a, b]$ takav da funkcija mijenja znak na tom intervalu (što u slučaju neprekidnosti funkcije garantira postojanje barem jedne nule u tom intervalu). Pri tome, treba voditi računa da "lovljenje" ne smije da "padne" ukoliko se u toku "lovljenja" slučajno izađe izvan domena funkcije, odnosno algoritam treba da može da se "oporavi" od takvih situacija. Prvi parametar funkcije "`BracketRoot`" je funkcija f čija se nula ograđuje. Funkcija je napisana kao generička, da bi se omogućilo da prvi parametar može biti ne samo funkcija, nego i bilo šta što se može koristiti poput funkcije, npr. neki funkcijski objekat (alternativno bi prvi parametar mogao biti i polimorfni funkcijski omotač tipa "`std::function<double(double)>`", ali su generičke tehnike za ovu svrhu efikasnije). Drugi parametar je početna vrijednost od koje započinje pretraga. Treći i četvrti parametar predstavljaju promjenljive u koje će se smjestiti nađene vrijednosti ukoliko je pretraga bila uspješna. Ostali parametri su opcionalni, odnosno imaju podrazumijevane vrijednosti. Peti parametar je početna vrijednost koraka pretrage (podrazumijevano 10^{-5}), šesti parametar je maksimalna vrijednost koraka pretrage (po modulu), pri čemu se pretraga prekida ukoliko se premaši ova vrijednost (podrazumijevano 10^{10}), dok je posljednji parametar faktor širenja koraka pretrage (podrazumijevano 1.4). Ukoliko pretraga ne uspije, prije nego što se definitivno odustane, treba ponoviti pokušaj, samo ovaj put sa negiranom početnom vrijednošću koraka. Ukoliko pretraga uspije (bilo u prvom ili drugom pokušaju), funkcija kao rezultat vraća "**`true`**", pri čemu će nađene vrijednosti a i b biti smještene u treći i četvrti parametar funkcije. Pri tome treba paziti da vraćene vrijednosti budu takve da je uvijek $a < b$. Ukoliko pretraga ne uspije ni nakon drugog pokušaja (što se najprije može desiti ukoliko funkcija uopće nema nula), funkcija kao rezultat vraća "**`false`**", a vrijednosti trećeg i četvrtog parametra tada su nedefinirane. Zadane vrijednosti parametara "`hinit`", "`hmax`" i "`lambda`" moraju biti pozitivne, u suprotnom treba baciti izuzetak tipa "`domain_error`" uz prateći tekst "Invalid parameters".

Sljedeća funkcija je "`RegulaFalsiSolve`", koja ima sljedeći prototip:

```
template <typename FunType>
  double RegulaFalsiSolve(FunType f, double a, double b,
    RegulaFalsiMode mode = Slavic, double eps = 1e-10, int maxiter = 100);
```

Pri tome je "`RegulaFalsiMode`" pobrojani tip definiran na globalnom nivou kao

```
enum RegulaFalsiMode {Unmodified, Illinois, Slavic, IllinoisSlavic};
```

Ova funkcija nalazi nulu funkcije $f$koja se nalazi unutar intervala $[a, b]$, za koji se pretpostavlja da ograđuje nulu, koristeći razne varijante metoda lažnog položaja (regula falsi). Prvi parametar je funkcija čija se nula traži, i za njega vrijedi sve što i za prethodnu funkciju. Naredna dva parametra su granice intervala a i b. Normalno bi trebalo biti $a < b$, ali funkciji ne bi trebalo da smeta ukoliko to nije ispunjeno. Međutim, u slučaju da su $f(a)$ i $f(b)$ istog znaka, funkcija treba da baci izuzetak tipa "`range_error`" uz prateći tekst "Root must be bracketed". Četvrti parametar određuje varijantu metoda lažnog položaja koji će se koristiti. Ovisno koju vrijednost taj parametar ima, treba respektivno koristiti izvornu (nemodificiranu) verziju, zatim Illinois verziju, slijedi verzija sa Slavićevom modifikacijom korištenjem pomoćne funkcije $\varphi(x) = x/(1 + |x|)$, te kombinaciju Illinois verzije sa Slavićevom modifikacijom (podrazumijevano se koristi Slavićeva modifikacija). Peti parametar je gornja granica greške (pri čemu je podrazumijevana vrijednost 10^{-10}), dok je šesti parametar maksimalno dozvoljeni broj iteracija (podrazumijevano 100). Funkcija kao rezultat vraća nađenu vrijednost nule. U slučaju da se zadana tačnost ne ostvari u maksimalno dozvoljenom broju iteracija, funkcija treba da baci izuzetak tipa

"logic_error" uz prateći tekst "Given accuracy has not been achieved". Zadane vrijednosti parametara "eps" i "maxiter" moraju biti pozitivne, u suprotnom treba baciti izuzetak tipa "domain_error" uz prateći tekst "Invalid parameters".

Slična ovoj funkciji je i funkcija "RiddersSolve", koja ima sljedeći prototip:

```
template <typename FunType>
  double RiddersSolve(FunType f, double a, double b, double eps = 1e-10,
    int maxiter = 100);
```

Za ovu funkciju vrijedi sve isto kao i za prethodnu funkciju, osim što se umjesto raznih varijanti metoda lažnog položaja za nalaženje nule koristi Riddersov algoritam.

Sljedeća funkcija koju treba napraviti je "NewtonRaphsonSolve", koja ima sljedeći prototip:

```
template <typename FunType1, typename FunType2>
  double NewtonRaphsonSolve(FunType1 f, FunType2 fprim, double x0,
    double eps = 1e-10, double damping = 0, int maxiter = 100);
```

Ova funkcija nalazi nulu funkcije f, koja (za razliku od prethodnih funkcija) ne mora prethodno biti ograđena, koristeći standardni Newtonov (odnosno Newton-Raphsonov) algoritam, uz opcionalnu adaptaciju korektivnog koraka s ciljem da se obezbijedi konvergencija za širi skup početnih tačaka. Prvi parametar je funkcija čija se nula traži, i za njega vrijedi sve što i za prethodne funkcije. Sljedeći parametar je funkcija koja računa vrijednost izvoda funkcije f (na primjer, ukoliko prva funkcija računa vrijednost izraza $f(x) = x^2 + 5x + 6$, druga funkcija računa vrijednost izraza $f'(x) = 2x + 5$. Treći parametar je početna aproksimacija za nulu koja se traži, dok je četvrti i peti parametar tolerancija koju želimo postići (podrazumijevano 10^{-10}). Što se tiče parametra "damping", ukoliko je on nula, vrši se klasični nemodificirani (neprigušeni) Newtonov algoritam. Međutim, u slučaju da je ovaj parametar različit od nule, u slučaju potrebe treba kratiti korektivni korak u Newtonovom algoritmu dok se ne postigne da se vrijednost funkcije po modulu smanji (na način opisan na predavanjima), ili dok se ne vratimo ponovo unutar domena funkcije (u slučaju da smo "odlutali" izvan domena). U tom slučaju vrijednost ovog parametra predstavlja vrijednost s kojim se množi korektivni korak sve dok on ne postane prihvatljiv. Konačno, posljednji parametar predstavlja maksimalno dozvoljeni broj iteracija (podrazumijevano 100). U slučaju da se konvergencija uz zadanu toleranciju ne ostvari u maksimalno dozvoljenom broju iteracija, funkcija treba da baci izuzetak tipa "logic_error" uz prateći tekst "Convergence has not been achieved". Ukoliko ne vršimo adaptaciju korektivnog koraka, ista stvar treba da se desi i ukoliko "odlutamo" izvan domena funkcije, ili ukoliko se u toku računanja dogodi dijeljenje sa nulom (situacija kada je $f'(x) = 0$ u tekućoj tački),. Za parametre "eps" i "maxiter" vrijedi što i u prethodnim funkcijama. Izuzetak tipa "domain_error" uz prateći tekst "Invalid parameters" također treba baciti i u slučaju da je parametar "damping" negativan ili veći ili jednak od 1.

Naredne dvije funkcije su funkcije istog imena "PolyRoots", sa sljedećim prototipovima:

```
std::vector<std::complex<double>>PolyRoots(
  std::vector<std::complex<double>> coefficients, double eps = 1e-10,
  int maxiters = 100, int maxtrials = 10);

std::vector<std::complex<double>> PolyRoots(
  std::vector<double> coefficients, double eps = 1e-10,
  int maxiters = 100, int maxtrials = 10);
```

Ove dvije funkcije služe za nalaženja nula polinoma, a jedina razlika je u tome što prva funkcija dopušta da koeficijenti polinoma mogu biti i kompleksni brojevi, dok druga pretpostavlja da su koeficijenti isključivo realni brojevi. Ove funkcije nalaze nule metodom deflacije, pri čemu se za nalaženje svake pojedinačne nule koristi Laguerreov algoritam. Za slučaj kada koeficijenti mogu biti i kompleksni brojevi, redukcija se vrši za svaku nulu pojedinačno, tako da se u svakom koraku deflacije stepen polinoma spušta za 1. Za slučaj kada su koeficijenti isključivo realni brojevi, pri svakoj kompleksnoj nuli, redukcija se vrši istovremeno i za njen konjugovano kompleksni par, tako da se tada stepen polinoma spušta za 2, održavajući tako koeficijente polinoma realnim tokom cijelog procesa. Poliranje rješenja nije neophodno, ali se može uraditi za dodatni 1 poen. Prvi parametar je vektor koeficijenata polinoma (redom od slobodnog člana naviše). Ostali parametri su opcionalni. Drugi parametar je tražena tolerancija (podrazumijevano 10^{-10}), treći parametar je maksimalni broj iteracija koji je dopušten za nalaženje jedne nule (podrazumijevano 100), dok je četvrti parametar maksimalni

broj pokušaja koliko će se puta pokušati naći nula (uz različite početne vrijednosti) u slučaju da konvergencija u prethodnom pokušaju nije postignuta. Svi ovi parametri moraju biti pozitivni, u suprotnom treba baciti izuzetak tipa "`domain_error`" uz prateći tekst "Invalid parameters". Funkcija vraća kao rezultat vektor čiji su elementi nađene nule, sortiran u rastući poredak po realnim dijelovima rješenja, a zatim po imaginarnim dijelovima rješenja u slučaju jednakih realnih dijelova. Ukoliko se za bilo koju nulu konvergencija ne uspije uspostaviti u zadanom broju pokušaja, funkcija prekida rad uz bacanje izuzetka "Convergence has not been achieved".

Obavezno napišite i testni glavni program u kojem ćete testirati napisane funkcije. Testiranje obavite na više različitih funkcija za koje su Vam vrijednosti nula poznate, te polinoma čije su vam nule poznate (množenjem malih faktora lako možete konstruisati polinome sa poznatim nulama, a također za provjeru možete koristiti i odgovarajuće funkcije iz Julia programskog jezika). Ne zaboravite testirati i razne patološke slučajeve (situacija kada funkcija nema nula, situacija sa višestrukim nulama, itd.), uključujući i neke opisane slučajeve za koje je poznato da neki od navedenih algoritama (npr. Newtonov) neće konvergirati.

Zadatak 2 (1 poen)

U ovom zadatku, potrebno je napisati funkciju nazvanu "`FindMinimum`" za numeričko nalaženje lokalnih minimuma funkcija jedne realne promjenljive. Funkcija treba da ima sljedeći prototip:

```
template <typename FunType>
  double FindMinimum(FunType f, double x0, double eps = 1e-8,
    double hinit = 1e-5, double hmax = 1e10, double lambda = 1.4);
```

Za nalaženje minimuma koristi se kombinacija tehnike "lovljenja" sa algoritmom zlatnog presjeka. Prvi parametar je funkcija f čiji se minimum traži, i za nju vrijedi sve što i za prethodne funkcije. Sljedeća dva parametra predstavljaju tačku x_0 od koje započinje potraga i inicijalni korak potrage h. Potraga se vrši u smjeru u kojem funkcija opada. Dakle, ukoliko je $f(x_0 + h) < f(x_0)$, smjer potrage je korektan. Ukoliko je umjesto toga $f(x_0 - h) < f(x_0)$, pretragu treba vršiti u suprotnom smjeru, tj. sa negiranim h. Ukoliko ne vrijedi niti jedno od toga, tada već trojka $(x_0 + h, x_0, x_0 + h)$ ograđuje minimum, pa potragu ne treba ni raditi. Treba implementirati "naivnu" verziju potrage (bez pokušaja ubrzavanja pomoću aproksimacije parabolama i sličnih sofisticiranih trikova). Drugim riječima, potraga se vrši postepeno povećavajući korak sa faktorom zadanim parametrom "`lambda`", sve dok se ne nađu tri uzastopne tačke koje ograđuju minimum, ili dok korak ne premaši vrijednost zadanu parametrom "`hmax`". U posljednjem slučaju, treba baciti izuzetak tipa "`logic_error`" uz prateći tekst "Minimum has not found". Nakon što se ogradi minimum, prelazi se na njegovo precizno lociranje koristeći algoritam zlatnog presjeka, tačnije njegovu modificiranu verziju koja garantira konvergenciju ka minimumu uz pretpostavku da je minimum prethodno ograđen uz pomoć tri tačke (kao što ovdje i jeste). Postupak se završava kada se interval koji sadrži mimimum svede na dužinu manju od vrijednosti parametra "`eps`". Funkcija tada vraća središte intervala kao rezultat. Parametri "`eps`", "`hinit`", "`hmax`" i "`lambda`" moraju biti pozitivni, inače treba baciti izuzetak tipa "`domain_error`" uz prateći tekst "Invalid parameters".

Obavezno napišite i testni glavni program u kojem ćete testirati napisanu funkciju nad funkcijama čiji su Vam minimumi poznati. Ne zaboravite testirati i razne patološke slučajeve, poput situacija kada funkcija nema minimuma, itd.

Zadatak 3 (2 poena)

U ovom zadatku, potrebno je razviti funkciju "`RK4Integrator`", kao podršku numeričkom rješavanju običnih diferencijalnih jednačina prvog reda, kao i funkciju "`RK4SystemIntegrator`", kao podršku numeričkom rješavanju sistema običnih diferencijalnih jednačina prvog reda. Funkcija "`RK4Integrator`" implementira Runge-Kutta metod četvrog reda u dvije verzije (sa i bez adaptacije koraka), a ima sljedeći interfejs:

```
template <typename FunType>
  std::vector<std::pair<double, double>> RK4Integrator(FunType f, double x0,
    double y0, double xmax, double h, double eps = 1e-8, bool adaptive = false);
```

Pretpostavimo li da jednačina koju rješavamo ima oblik $y' = f(x, y)$, prvi parametar predstavlja upravo funkciju f. Za ovaj parametar vrijedi sve što i u prethodnim funkcijama, osim što je f funkcija koja ovisi od dva realna parametra. Naredna dva parametra predstavljaju vrijednosti x_0 i y_0 iz početnog

uvjeta $y(x_0) = y_0$, dok četvrti parametar predstavlja vrijednost x_{max} do koje se vrši simulacija. Peti parametar je fiksna vrijednost koraka (za režim rada bez adaptacije) odnosno početna vrijednost koraka (ukoliko se vrši adaptacija). Šesti parametar je dozvoljena tolerancija, dok sedmi parametar određuje da li će se vršiti automatska adaptacija koraka ili ne. Ukoliko se zahtjeva adaptacija koraka, koristi se tehnika polovljenja koraka. Funkcija kao rezultat vraća vektor uređenih parova (x_i, y_i) u svim tačkama u kojim je vršena simulacija. Pri tome, za slučaj kada se vrši adaptacija koraka, treba obezbijediti da u izlazni vektor obavezno uđe i par čija je prva koordinata tačno x_{max}. Konačno, treba podržati da se simulacija može izvoditi i "unazad", zadavanjem negativnog koraka h i vrijednosti x_{max} koja je manja od x_0 (ništa se ovdje suštinski ne mijenja osim nekih nejednakosti, npr. kriterija pod kojim se simulacija zaustavlja). U slučaju da je $x_{max} < x_0$ pri pozitivnom h, ili $x_{max} > x_0$ pri pozitivnom h, simulacija se odmah prekida, i izlazni vektor treba sadržavati samo početnu tačku (x_0, y_0).

Funkcija "`RK4SystemIntegrator`" implementira Runge-Kutta metod četvrog reda s fiksnim korakom, a ima sljedeći interfejs:

```
template <typename FunType>
  std::vector<std::pair<double, std::vector<double>>> RK4SystemIntegrator(
    FunType f, double x0, std::vector<double> y0, double xmax, double h);
```

Pretpostavimo da sistem jednačina koju rješavamo ima oblik

$$y_1' = f_1(x, y_1, y_2, \ldots, y_n)$$
$$y_2' = f_2(x, y_1, y_2, \ldots, y_n)$$
$$\vdots$$
$$y_n' = f_n(x, y_1, y_2, \ldots, y_n)$$

Kompaktno se ovaj sistem može napisati u obliku $\boldsymbol{y}' = \boldsymbol{f}(x, \boldsymbol{y})$, gdje je $\boldsymbol{y}$ vektor čije su komponente y_1, y_2, …, y_n, a $\boldsymbol{f}$ je vektorska funkcija čiji je rezultat vektor čije su komponente $f_1(x, \boldsymbol{y})$, $f_2(x, \boldsymbol{y})$, …, $f_n(x, \boldsymbol{y})$. Prvi parametar funkcije "`RK4SystemIntegrator`" je upravo vektorska funkcija $\boldsymbol{f}$, izvedena kao funkcija od dva argumenta, pri čemu je prvi argument realan broj x, a drugi vektor realnih brojeva $\boldsymbol{y}$, a koja kao rezultat vraća vektor realnih brojeva. Na primjer, ukoliko želimo riješiti sistem

$$y_1' = 2\,y_1 + 3\,y_2 + x$$
$$y_2' = y_1 - 2\,y_2 + 1$$

kao prvi parametar funkciji "`RK4SystemIntegrator`" možemo poslati imenovanu funkciju "`f`" definiranu kodom

```
std::vector<double> f(double x, std::vector<double> y) {
  return {2 * y[0] + 3 * y[1] + x, y[0] - 2 * y[1] + 1};
}
```

ili anonimnu lambda funkciju definiranu izrazom

```
[](double x, std::vector<double> y) -> std::vector<double> {
  return {2 * y[0] + 3 * y[1] + x, y[0] - 2 * y[1] + 1};
}
```

Ovaj parametar je izveden tako da može biti ne samo funkcija, nego i bilo šta što se može koristiti poput funkcije, npr. neki funkcijski objekat (alternativno bi prvi parametar mogao biti i polimorfni funkcijski omotač tipa "`std::function<std::vector<double>(double,std::vector<double>)>`", ali su generičke tehnike za ovu svrhu efikasnije). Naredna dva parametra "`x0`" i "`y0`" predstavljaju vrijednosti x_0 i $\boldsymbol{y}_0$ iz (vektorskog) početnog uvjeta $\boldsymbol{y}(x_0) = \boldsymbol{y}_0$. Vektor "`y0`" mora imati isto onoliko elemenata koliko iznosi veličina vektora vraćenog iz vektorske funkcije zadane prvim parametrom, u suprotnom treba baciti izuzetak tipa "`range_error`" uz prateći tekst "Incompatible formats". Četvrti parametar predstavlja vrijednost x_{max} do koje se vrši simulacija, dok je peti parametar je vrijednost koraka integracije. Funkcija kao rezultat vraća vektor uređenih parova $(x_i, \boldsymbol{y}_i)$ u svim tačkama u kojim je vršena simulacija (obratite pažnju da je druga komponenta ovih uređenih parova također vektor, koji sadrži vrijednosti svih funkcija koje učestvuju u sistemu). Treba podržati da se simulacija može izvoditi i "unazad", zadavanjem negativnog koraka h i vrijednosti x_{max} koja je manja od x_0. U slučaju da je $x_{max} < x_0$ pri

pozitivnom h, ili $x_{max} > x_0$ pri pozitivnom h, simulacija se odmah prekida, i izlazni vektor treba sadržavati samo početnu tačku $(x_0, \boldsymbol{y}_0)$.

Obavezno napišite i testni glavni program u kojem ćete testirati napisane funkcije koristeći kao test diferencijalne jednačine odnosno sisteme diferencijalnih jednačina čija su Vam rješenja poznata (ograničite se na jednostavne diferencijalne jednačine poput $y' = 3\,x + 2\,y$).

Zadatak 4 (1 poen)

U ovom zadatku potrebno je razviti funkcije za podršku kompresiji s gubicima sporopromjenljivih sekvcenci, zasnovanu na diskretnoj Fourierovoj transformaciji, te za približnu rekonstrukciju izvorne sekvence iz kompresovane sekvence. Za tu svrhu, predviđene su dvije funkcije nazvane "`LossyCompress`" odnosno "`LossyDecompress`", sa sljedećim prototipovima:

```
std::vector<double> LossyCompress(std::vector<double> data, int new_size);
std::vector<double> LossyDecompress(std::vector<double> compressed);
```

Funkcija "`LossyCompress`" obavlja kompresiju s gubicima sekvence pohranjene u prvom parametru na dužinu zadanu drugim parametrom, i vraća kao rezultat kompresovanu sekvencu. Ovaj parametar (nazovimo ga M) mora biti veći od 1 i manji ili jednak od dužine sekvence, inače treba baciti izuzetak tipa "`range_error`" uz prateći tekst "Bad new size". Kompresija se vrši tako što se na izvornu sekvencu primijeni diskretna kosinusna transformacija, a zatim se iz transformirane sekvence zadrži samo $\mathrm{M}-1$ prvih članova, a kao M-ti član stavlja se dužina izvorne sekvence (ova informacija će nam trebati za potrebe dekompresije). Što je M manji, treba očekivati i veće gubitke, odnosno veća izobličenja sekvence nakon obavljene dekompresije. Diskretnu kosinusnu transformaciju treba računati svođenjem na diskretnu Fourierovu transformaciju, koju opet treba računati pomoću nekog od FFT algoritama. Zbog ograničenja FFT algoritama koje smo razmatrali, dužina sekvence koja se kompresuje mora biti neki stepen broja 2, inače treba baciti izuzetak tipa "`range_error`" uz prateći tekst "Data size must be a power of two".

Funkcija "`LossyDecompress`" obavlja dekompresiju kompresovane sekvence zadane parametrom, odnosno pokušava da izvrši aproksimativnu rekonstrukciju izvorne sekvence. Prvo se iz posljednjeg elementa kompresovane sekvence čita pohranjena informacija o dužini izvorne sekvence (nazovimo je N). Ukoliko N nije prirodan broj koji je stepen dvojke, ili ukoliko je N manji od dužine kompresovane sekvence, treba baciti izuzetak tipa "`logic_error`" uz prateći tekst "Bad compressed sequence". Nakon čitanja N, kompresovanu sekvencu treba dopuniti nulama do dužine N (pohranjenu informaciju o dužini sekvence pri tome također treba zamijeniti nulom), nakon čega na tako dopunjenu sekvencu treba primijeniti inverznu kosinusnu transformaciju (koju treba računati svođenjem na inverznu Fourierovu transformaciju u primjenom nekog od FFT algoritama), čime se dobija dekompresovana sekvenca (tj. aproksimacija izvorne sekvence).

Obavezno napišite i testni glavni program u kojem ćete testirati napisane funkcije. Za tu svrhu, možete iskoristiti činjenicu da se recimo sekvenca koja predstavlja uzorke sinusne funkcije uzete u ravnomjernim intervalima na jednom periodu može vrlo uspješno kompresovati na vrlo malu dužinu.

NAPOMENA:

Naučite se da testirate programe sami, nemojte čekati na autotestove da vidite da li Vam je nešto ispravno ili ne. Pregledanje zadaće će se vršiti kombinacijom autotestova i ručnog pregledanja. Ono što nije testirano smatraće se da nije urađeno, te se te elemente neće dobiti nikakvi bodovi!