Numerične metode v fiziki: domače naloge

Vaje iz predmeta Numerične metode v fiziki

Igor Grešovnik

Revision 3, February 2015.

Revision 0: January 2008.

Vsebina:

1 NALOGA 1A - ARITMETIČNA IN GEOMETRIJSKA ZAPOREDJA

Napiši program za računanje delnih vsot aritmetičnih in geometrijskih zaporedij:

$$\sum_{i=0}^{n} a_0 + i\,d = \frac{a_0 + a_n}{2}(n+1)$$

$$\sum_{i=0}^{n} a_0\, r^{\,i} = a_0 \frac{r^{n+1} - 1}{r - 1}$$

Program naj izpiše vse delne vsote do danega n, izračunane s seštevanjem členov in z neposredno formulo.

Izpiši *člene in delne vsote* od 1 do 20 aritmetičnega zaporedja s podatki

$a_0 = 5$

$d = 0{,}5$

in geometrijskega zaporedja s podatki

$a = 2$

$r = 0{,}5$

2 NALOGA 1B - NUMERIČNA INTEGRACIJA

Implementiraj numerično integracijo realne funkcije ene spremenljivke po Simpsonovi formuli.

Simpsonovo in trapezno metodo uporabi za izračun integrala

$$\int_0^{2\pi} \sin(x) + e^x \,.$$

in opazuj, kako se povečuje natančnost pri obeh metodah, ko večamo število intervalov.

Namig:

Enostaven program za numerično integriranje je implementiran v projektu 01integration\ v direktoriju csharp/gresovnik na študentskem FTP računu. Do projekta najlažje pridete tako, da v Visual Studio C# odprete datoteko test_solution.sln iz tega direktorija, ki si ga najprej presnamete na lokalni disk.

Navedeni program uporablja metode in iz projekta ***00library*** (definirane so v datoteki integration.cs). Metode imajo med svojimi argumenti delegata ***RealFunction***, ki je definiran v datoteki 0definitions.cs projekta 00library.

3 NALOGA 1C - POSPEŠEVANJE AVTOMOBILA

Reši nalogo o pospeševanju avtomobila na strain:
http://www2.ijs.si/~zidansek/tekst1.html

Pri reševanju uporabi trapezno in Simpsonovo formulo. Funkcij ni potrebno risati, lahko jih samo tabeliraš in skopiraš tabele v poročilo.

4 DODATNA NALOGA - INTEGRACIJA

To je dodatna naloga za študente, ki niso znali rešiti naloge. Ti študentje naj najprej rešijo to nalogo, nato pa še redne naloge.

Naloga:
Izračunaj določeni integral v mejah od 0 do 1 funkcije

$$f(x) = x\, e^x\ .$$

Nedoločeni integral funkcije je

$$I(x) = x\, e^x - e^x + C,$$

Kjer je C nedoločena konstanta.

Integral izračunaj po trapezni metodi v mejah od s 4, 16, 32, 64 in 128 podintervali in ga primerjaj z analidično rešitvijo po spodnjih navodilih. Za integriranje po trapezni metodi naredite metodo, ki **ne shranjuje** vrednosti funkcije v vseh točkah, in rezultate s svojo metodo primerjajte z rezultati z metodo, ki je definirana v projektu gresovnik/00library v datoteki integration.cs.

1. Naredite kopijo direktorija csharp/template_solution in ga poimenujte s ”<priimek>.<ime>.1”
   a. v imenu nadomestite <priimek> in <ime> z vašim priimkom in imenom
2. V tem direktoriju preimenujte datoteko template_solution.sln v ”<priimek>.<ime>.sln”. To bo rešitev, v kateri boste naredili nalogo.
3. Odprite rešitev v Visual Studiu in v rešitev dodajte nov projekt – konzolno aplikacijo z imenom »domaca_naloga_01dodatna_<priimek>«
4. Datoteko v projektu, kjer imate metodo Main, preimenujte v Program_01.cs.
5. **Imenski prostor** (angleško »namespace«) v tej datoteki preimenujte v **Naloga01**.
6. V tej datoteki vstavite kak enostaven izpis na konzolo in poženite program, da vidite, če pravilno deluje.
7. V rešitev vključite obstoječ projekt, ki je v csharp/gresovnik/00library.
8. V projektu za nalogo naredite novo **datoteko** v C# z imanom **MyIntegration.cs**, v njej pa definirajte **razred MyIntegration**, ki naj vsebuje **statično metodo**

`MyIntegralTrapezoidal`. Razred naj bo definiran **v imenskem prostoru** `MyLib`! Metoda naj ima isti podpis (torej argumente istega tipa) kot prva metoda `IntegralTrapezoidal` definirana v 00integration.cs v projektu 00library, ki ste ga prej vključili. Poskrbeti boste morali, da bodo uporabljeni razredi in drugi tipi vidni v vaši izvorni datoteki! Metodo boste implementirali pozneje.

9. Naredite novo datoteko v C# z imenom **definitions.cs**! V njej v imenskem prostoru `MyLib` definirajte **razred** `Definitions`, ki naj vsebuje **statično metodo** `Function1Dodatna` , katere podpis ustreza delegatu RealFunction (definirana v projektu 00library v datoteki 0definitions.cs). Metodo implementirajte tako, da vrne vrednost funkcije, ki jo morate integrirati! . Poskrbeti boste morali, da bodo uporabljeni razredi in drugi tipi vidni v vaši izvorni datoteki!
10. V razredu Definitions implementirajte še statično metodo `Integral_function1Dodatna`. Funkcijo implementirajte tako, da vrne nedoločen integral (brez nedoločene konstante) funkcije, ki jo morate integrirati. Analitično vrednost določenega integrala za primerjavo lahko potem izračunate tako, da kličete to funkcijo pri zgornji meji integral in od nje odštejete vrednost te funkcije pri spodnji meji.
11. V razredu projekta za rešitev naloge, v katerem je metoda Main, definirajte metodo `void Naloga1Dodatna_IntegrationFromLibrary(void)`. V tej metodi implementirajte integriranje z uporabo metode za integriranje `IntegralTrapezoidal` , ki je definirana v integration.cs v projektu 00library. Integriranje naj bo po navodilih iz naloge! Primerjajte izračunane integrale z analizično rešitvijo! Na začetku metode vstavite izpis, ki bo na dobro viden način povedal, katero metodo ste klicali (to bo prišlo prav pozneje, ko boste klicali več podobnih metod). Tudi tu boste morali poskrbeti za to, da bodo uporabljeni tipi vidni v datoteki, v kateri programirate. Metodo kličite iz metode Main in poženite program! Če boste imeli kakšne napake in bo videti, da rezultati niso pravilni, metodo popravite.
    a. Za izračun analitične vrednosti uporabite metodo `Integral_function1Dodatna`, ki vrne nedoločen integral funkcije, ki jo integrirate! V metodi definirajte dve spremenljivki tipa **RealFunction** (to je delegat, ki je definiran v datoteki 0definitions.cs v datoteki 00library). Eni spremenljivki priredite metodo `Function1Dodatna` , drudi pa metodo `Integral_function1Dodatna` iz razreda `Definitions` v datoteki definitions.cs. Pri naslavljanju teh metod morate navesti ime razreda (ki ga s piko ločite od imena metode), ker sta metodi statični. Poskrbeti morate tudi za to, da je razred viden.
12. Implementirajte medoto `MyIntegralTrapezoidal` v razredu MyIntegration.
13. V razredu projekta za rešitev naloge, v katerem je metoda Main, definirajte metodo `void Naloga1Dodatna_MyIntegration(void)`. V tej metodi implementirajte integriranje z uporabo svoje svoje metode `MyIntegralTrapezoidal`, metodo pa naredite podobno kot tisto za uporabo integriranja iz knjižničnega projekta 00lbrary. Metodo prav tako kličite iz metode Main. Klic te metode samo dodajte za klicem prejšnje metode in prejšnji klic zakomentirajte!
14. V razredu projekta za rešitev naloge, v katerem je metoda Main, definirajte še metodo `void Naloga1Dodatna (void)`. V tej metodi implementirajte integriranje z uporabo obeh metod za integracijo (metode iz knjižnice in svoje lastne) tako, da pri vsakem številu podintervalov tudi primerjate rešitev po eni in drugi metodi ter analitično rešitev. Tudi klic te metode dodajte v metodo Main in tam zakomentirajte prejšnja dva klica! V metodi Main boste tako vedno lahko odkomentirali klice metod, ki jih želite izvajati, in zakomentirali ostale.

15. Ko bodo stvari delovale, preoblikujte metodo `Naloga1Dodatna_MyIntegration` tako, da jo boste lahko uporabili za poljubno funkcijo, pri kateri poznate analitičen izraz za nedoločen integral. Metodo skopirajte in jo preimenujte v `Naloga1Dodatna_MyIntegration_Generic`. Metodi (ki je bila prej brez argumentov) dodajte dva argumenta tipa RealFunction. Prvi argument naj bo metoda, ki vrne vrednost funkcije, ki jo integrirate. Drugi argument naj bo metoda, ki vrne vrednost nedoločenega integrala te funkcije. To bo zelo enostavno narediti, ker boste samo obe lokalni delegatski spremenljivki premaknili iz telesa metode med argumente. Spremenljivki boste morali seveda prirediti pred klicem metode. Tudi to metodo kličite v metodi Main, pred klicem pa definitajte obe delegatski spremenljivki, ki ju boste podali kot argumenta, in jima priredite ustrezne vrednosti.
16. Sedaj nalogo uporabite za integracijo druge funkcije in sicer $f(x) = x\,e^x$. Nedoločen integral te funkcije je $I(x) = x + e^x + C$. Nalogo rešite enostavno tako, da **v razredu Definitions** definirate metodi za izračun vrednosti funkcije in vrednosti njenega nedoločenega integrala, ter v metodi Main kličete metodo `Naloga1Dodatna_MyIntegration_Generic`, ki ji preko argumentov prenesete delegate tipa RealFunction, ki sta inicializirana na ti dve novo definirani metodi. Metodi poimenujte `Function1Dodatna2` in `Integral_function1Dodatna2`.

5 SPLOŠNA NAVODILA

Za vsako vajo napiši kratko in razumljivo poročilo v urejevalniku besedil (npr. MS Word).

Programe in projekte v C# umesti v dogovorjeno direktorijsko strukturo.

Nekaj tekstov in nalog iz numeričnih metod je na naslednjih straneh:
http://www2.ijs.si/~zidansek/num2003.html

Dodatno gradivo najdeš na FTP strežniku z naslednjimi podatki za dostop:
ftp://164.8.24.171
port: 2222
username: student
password: student

5.1 Namig za kopiranje iz konzole

Kopiranje v konzoli je možno le, če to nastaviš pri lastnostih konzole. Klikneš na kono konzole v levem zgornjem kotu okna, izbereš Properties/Options/ in pod Edit Options vse odkljukaš.

Kljub temu je kopiranje lahko nerodno, če je teksta za več vrstic ali so v konzoli lomljene vrstice. Problem lahko rešiš tudi tako, da poženeš program, ki izpisuje v konzolo, in mu preusmeriš standardni izhod (ki je prednastavljen na konzolo) v izbrano datoteko. To narediš tako, da program poženeš na naslednji način:

program.exe argumenti... > out.txt

Tu je predpostavljeno ime programa *program.exe*. Ko prevedeš projekt v C#, katerega rezultat je program, lahko ta program najdeš nekje znotraj projektnega direktorija, tipično v direktoriju /bin/debug relativno glede na projektni direktorij.

Z opisanim načinom so lahko problemi, kadar program bere podatke, ki jih vstavi uporabnik preko konzole. V takšnem primeru je najbolje v programu izključiti vsa interaktivna branja in potem pognati program s preusmerjenim standardnim izhodom.

6 NALOGA 2A – RAZREDI V C#

V rešitvi *test_solution* v direktoriju *csharp* je v projektu *00library* datoteka *ClassComplex.cs*. V tej datoteki je definicija razreda *Complex*, ki predstavlja kompleksna števila. V komentarjih so pojasnila o relevantnih konceptih programskega jezika. Razred Complex vsebuje statično metodo *Example()*, ki demonstrira uporabo definiranega razreda. Metodo lahko kličeš v projektu *test_console*.

Po zgledu razreda *Complex* definiraj razred *Cvector*, ki predstavlja vektor kompleksnih števil. V razredu naj bodo definirane operacije seštevanja in odštevanja vektorja, skalarni produkt dveh vektorjev (upoštevaj konjugirano simetričnost skalarnega produkta), množenje vektorja s kompleksnim ali realnim skalarjem, ter indeksna operacija za dostop do komponent vektorja. Indeksi komponent naj tečejo od 0 naprej. Vektor naj ima lastnost *Dim*, ki vrne dimenzijo vektorja (vendar **se dimenzija ne da tudi nastaviti** preko te lastnosti).

Tip naj vsebuje konstruktor, katerega element je dimenzija števila, pri konstrukciji pa naj bodo vse komponente inicializirane na (0 + 0 *i*). Za vajo poleg tega definiraj še konstruktor, ki mu podaš dimenzijo in eno kompleksno število, na katerega se postavijo vse komponente vektorja. Komponente naj se kopirajo, vsako posebej torej alociraš z izrazom new Complex(...), kjer v npr. v konstruktorju podaš realni in kompleksni del kompleksnega števila, katerega kopijo tvoriš, dobiš ju z lastnostmi *Re* in *Im*.

Nasveti za izdelavo te naloge:

Še enkrat si poglej razred *Complex* v datoteki ClassComplex.cs. Pozorno preberi vse komentarje. Dobro je, da poskusiš nekajkrat izvesti funkcijo *Complex.Example()* (tako, da metodo kličeš iz metode *Main* v nekem projektu). Funkcijo lahko spreminjaš in opazuješ, kako razred *Complex* deluje.

To, da se dimenzije vektorja ne da nastaviti drugače, lahko dosežeš tako, da definiraš lastnost *Dim*, v kateri definiraš blok *set* kot *private*. Na ta način je blok set lastnosti *Dim* viden le znotraj razreda in lahko lastnosti *Dim* le znotraj razreda prirediš vrednost. V tem primeru bi za podporo lastnosti *Dim* definiral še privatno spremenljivko tipa int, v kateri je dejansko shranjena dimenzija (lastnosti – angleško "*properties*" so sintaktični konstrukti, ki delujejo v tem smislu bolj kot metode in sami po sebi ničesar ne hranijo). V konstrukrorju v tem primeru lahko priredišvrednost lastnost *Dim* (zaradi konsistentnosti jo tudi moraš nastaviti), saj je konstruktor del razreda in mu je zato viden tudi blok *get* lastnosti *Dim*, ki je definiran kot *private*.

Še boljša možnost je, da v *Dim* sploh ne definiraš bloka *set*, blok *get* pa naj vrne kar trenutno dimenzijo tabele, v kateri so shranjene komponente. V tem primeru ni treba posebej skrbeti za konsistentnost lastnosti *Dim*, ker ta vedno vrne dejansko število komponent vektorja. Če tabelo komponent imenuješ _components, dobiš dimenzijo tabele (število elementov, ki jih vsebuje) z izrazom *_components.Length*.

Za primer, kako se definirajo lastnosti, poglej lastnosti *Complex.Re* in *Complex.fi* (slednja nima definiranega bloka *set* in se torej ne da neposredno nastaviti).

Komponente vektorja naj bodo shranjene v tabeli kompleksnih števil, ki ni vidna navzven (definirana bo kot *private* ali *protected*). Dostop do posameznih komponent bo tako možen samo preko indeksnega operatorja.

Tabelo objektov tipa *Complex* definiraš na naslednji način:

```
private Complex[] _components;
```

Prostor za tabelo alociraš tako (to je obvezno, preden začneš v tabelo shranjevati elemente):

```
_components = new Complex[dim];
```

Tabelo komponent alociraš v konstruktorju, ki mu podaš dimenzijo vektorja (in s tem dimenzijo tabele komponent). S to dimenzijo (argument *dim* v zgornji vrstici tabele) alociraš tabelo. Ne pozabi, da indeksi tabele tečejo od 0 do *dim*-1! V konstruktorju naj se vsa števila postavijo na (0 + 0 *i*). Lahko bi definiral tudi konstruktor, ki mu podaš tabelo kompleksnih števil.

V razredu potrebuješ še način za nastavljanje in dostopanje do posameznih komponent. V ta namen definiraš *indeksni operator*. Primer definicije indeksnega operatorja si poglej v razredu *Complex*. Skica definicije je

```
public Complex this[int index] { get{ … } set { …  } }
```

V bloku *get* uporabiš besedo *return*, da vrneš zahtevano komponento vektorja, v bloku *set* pa preko preddefinirane spremenljivke *value* (ki je istega tipa kot indeksni operator, v tem primeru tipe Complex) dobiš vrednost, ki jo priredíš določeni komponenti.

Pri prirejanju komponent (blok *set* v definiciji indeksnega operatorja) ne priredi le reference, ampak kompleksno število kopiraj. To narediš tako, da narediš novo kompleksno število z operatorjem *new*, kjer v oklepaju navedeš komponente kompleksnega števila, ki ga kopiraš. To lahko dosežeš tudi z uporabo konstruktorja, ki ima za argument kompleksno število, npr.

```
_components[index] = new Complex(value);
```

7 NALOGA 2B – NUMERIČNO ODVAJANJE

Implementiraj metodi za prve in druge numerične odvode funkcije realne spremenljivke.

Metodi naj imata naslednje parametre:

- funkcija, ki jo odvajamo, v oblki delegata *RealFunction*, ki je definiran v datoteki *0definitions.cs* v projektu *00library*.
- vrednost argumenta funkcije, pri kateri iščemo odvod

- velikost koraka pri odvajanju

Delegat *RealFunction* je definiran na naslednji način:

```
public delegate double RealFunction(double x);
```

Funkcijo uporabi za odvajanje naslednjih funkcij:

$$f_1(x) = \sin(x) + e^x$$
$$f_2(x) = \left(1 + 10^{-6}\left(r - \frac{1}{2}\right)\right) f_1(x)\ ,$$

kjer je r v formuli za funkcijo $f_2(x)$ enakomerno porazdeljeno naključno število na intervalu [0,1]. Odvode teh funkcij računaj v 20 ekvidistančnih točkah na intervalu $[0, 2\pi]$. Za računanje uporabi korake h = 0,1, 10^{-2}, 10^{-4}, 10^{-6} in 10^{-8}. Primerjaj numerično izračunane vrednosti z analitičnimi.

Za generacijo enakomerno porazdeljenih naključnih števil na intervalu [0,1] lahko uporabiš metodo *NextDouble()* razreda *Random*:

```
Random rnd = new Random();
...
double x = rnd.NextDouble();
```

Opozorilo:
Funkcija $f_2(x)$ je ekvivalentna funkciji $f_1(x)$, ki ji je dodan beli šum z amplitudo pol milijonine vrednosti funkcije, zaradi tega sta analitični prvi in drugi odvod funkcije $f_2(x)$ enaka kot pri $f_1(x)$. Pri dodatku naključnega člena v definiciji funkcije $f_2(x)$ je treba paziti, da se pri vsakem izračunu funkcije generira novo naključno število. Programski generatorji naključnih števil so deterministični in generirajo vedno isto zaporedje števil po tem, ko jih inicializirano na določen način. Če bi pred vsakim klicem metode *NextDouble()* s klicem *new Random()* na novo generilali objekt, na katerem metodo kličemo, bi vsakič dobili začetno število tega zaporedja, torej vedno isto število. Zato je ključno, da objekt razreda *Random* naredimo z operatorjem *new* le enkrat in potem vsakič kličemo *NextDouble()* na istem objektu. Enostaven način za dosego tega je, da generator naključnih števil definiramo kot javno statično spremenljivko v nekem javnem razredu in ga tam tudi inicializiramo:

```
...
public class MyClass
{
      ...
      public static Random rnd = new Random();
      ...
}
```

Za numerično odvajanje uporabi formuli

$$f'(x_0) = \frac{1}{2h}\left(f(x_0+h) - f(x_0-h)\right) - \frac{h^2}{6} f^{(3)}(\xi),$$

$$f''(x_0) = \frac{1}{h^2}\left(f(x_0-h) - 2f(x_0) + f(x_0+h)\right) - \frac{h^2}{12} f^{(4)}(\xi);\ ,$$

$$x_0 - h < \xi < x_0 + h$$

kjer je *h* korak numeričnega odvajanja.

8 NALOGA 2C – REŠEVANJE NELINEARNIH ENAČB ENE SPREMENLJIVKE

Implementiraj metodi za dva postopka reševanja nelinearnih enačb - z Newtonovo metodo in nato še ali s sekantno metodo ali z bisekcijo.

Kriterij za ustavitev iteracije naj bo vezan na absolutno vrednost funkcije. Enačba naj bo podana v standardni obliki

$$f(x) = 0\ .$$

Pri Newtonovi metodi naj bodo parametri funkcija $f(x)$ in njen odvod $f'(x)$ (parametra naj bosta oba tipa *RealFunction*), začetna vrednost neodvisne spremenljivke in toleranca, pod katero mora pasti absolutna vrednost funkcije *f* (toleranca mora biti pozitivno število, v metodah je to potrebno preveriti in vreči izjemo, če ta pogoj ni izpolnjen).

Pri sekantni metodi naj bodo parametri funkcija $f(x)$, izhodiščni vrednosti neodvisne spremenljivke in toleranca, pod katero mora pasti absolutna vrednost funkcije *f*. Funkcijski parameter naj bo tipa *RealFunction*. Metoda naj preveri, ali sta vrednosti funkcije v izhodiščnih točkah nasprotno predznačeni, in naj vrže izjemo, če nista.

Pri bisekciji naj bodo parametri funkcija $f(x)$, izhodiščni vrednosti neodvisne spremenljivke in toleranca, pod katero mora pasti absolutna vrednost funkcije *f*. Metoda naj tudi tu preveri, ali sta vrednosti funkcije v izhodiščnih točkah nasprotno predznačeni, in naj vrže izjemo, če nista. Uporabiš lahko različne pristope za iskanje dveh začetnih točk, v katerih je funkcija $f(x)$ različno predznačena. Lahko na primer implementiraš metodo, ki izpiše vrednosti funkcije v 20 ali 30 ekvidistančnih točkah na danem intervalu in s pomočjo te metode na roke (s premikanjem krajišč intervala) poiščeš dve takšni točki. Do dveh primernih točk lahko prideš tudi z analizo grafa funkcije, ali pa razviješ poseben algoritem za iskanje takšnih točk.

Delegatski tip *RealFunction* je definiran v *csharp\gresovnik\00library\0definitions.cs*. Pri sekantni metodi in bisekciji Priporočljivo je, da pri vsaki od metod uvedeš še dodatni parameter, ki določa največje dovoljeno število iteracij. Iteracijski postopek naj se ustavi, če je preseženo največje dovoljeno število iteracij, in sicer ne glede na to, ali je absolutna vrednost funkcije, katere ničlo iščemo, že padla pod predpisano toleranco ali ne.

Metodi v prvem delu naloge uporabi pri reševanju naslednjih enačb:

$$\begin{aligned} e^x &= 10 \\ e^x &= -x \\ x^7 &= 10^{-8} \end{aligned} \quad .$$

V vseh primerih pri sekantni metodi uporabi izhodiščni vrednosti $x_1 = -1$ in $x_2 = 3$, pri Newtonovi metodi pa začetni približek $x_0 = 3$.

V drugem delu naloge rešuj z Newtonovo metodo razred enačb

$$x^7 = t$$

za 10 vrednosti *t*, ki so enakomerno razdeljene na intervalu $[0,3]$. Začetni približek naj bo vedno $x_0 = 1/2$.

9 NALOGA 2D – MATRIKE IN REŠEVANJE SISTEMA LINEARNIH ENAČB

V

csharp\gresovnik\00library\ClassMatrixGeneric.cs

in

csharp\gresovnik\00library\ClassVectorGeneric.cs

so definirani razredi, ki predstavljajo matrike in vektorje.

Najprej dopolni definicije teh razredov za računanje produktov matrik in produktov matrik z vektorji. Nato implementiraj še reševajnje sistemov linearnih enačb z Gaussovo eliminacijsko metodo.

Gaussovo eliminacijsko metodo uporabi pri reševanju sistema enačb

$$\mathbf{A}\,\mathbf{x} = \begin{aligned} x_1 + x_2/2 + x_3/3 &= 32 \\ x_1/2 + x_2/3 + x_3/4 &= 22 \\ x_1/3 + x_2/4 + x_3/5 &= 17 \end{aligned} = \mathbf{b} \quad .$$

Po rešitvi sistema enačb izračunaj normo $\left\| \mathbf{A}\,\tilde{\mathbf{x}} - \mathbf{b} \right\|_2$, kjer je $\tilde{\mathbf{x}}$ izračunana rešitev.

10SPLOŠNA NAVODILA

Za vsako napiši kratko in razumljivo poročilo v urejevalniku besedil (npr. MS Word).

Programe in projekte v C# umesti v dogovorjeno direktorijsko strukturo.

Nekaj tekstov in nalog iz numeričnih metod je na naslednjih straneh:
http://www2.ijs.si/~zidansek/num2003.html

Dodatno gradivo najdeš na FTP strežniku z naslednjimi podatki za dostop:
ftp://164.8.24.171
port: 2222
username: student
password: student

10.1 Namig - zapisovanje rezultatov v datoteko

V datoteko se zapisujejo rezultati podobno kot v konzolo, za to lahko uporabiš razred StreamWriter.

Primer:

```
static void FileTest()
{
    StreamWriter sw = new StreamWriter("c:\\temp\\test.txt", false  /* append */);
    sw.WriteLine("Zapis v testno datoteko... ");
    sw.WriteLine("  Cas: " + DateTime.Now.ToString());
    sw.WriteLine("  Pi ~ " + Math.PI.ToString());
    sw.WriteLine("Konec zapisa. ");
    sw.Close();
}
```

10.1Namig – predstavitev vektorjev in matrik s tabelami

Za predstavitev vektorjev pogosto uporabljamo enodimenzionalne tabele števil, za predstavitev matrik pa dvodimenzionalne tabele oziroma tabele tabel števil (“jagged arrays” v angleščini). V jeziku C# se tabele obnašajo kot objekti in imajo zapisano svojo dolžino v lastnosti *Length*.

Primer:

```
    Console.WriteLine(Environment.NewLine + "Matrix - vector multiplication example:" +
Environment.NewLine);
    double[] x, b; // two 1D arrays to represent vectors
    double[][] A; // 2D array to represent a matrix (note double square brackets [][])
    int d1 = 2, d2 = 3; // dimensions of a rectangular matrix
    // Allocate vector b:
    x = new double[d2];
    // Allocate the matrix:
    A = new double[d1][];  // allocate outer-most array (array of arrays of double)
    for (int i = 0; i < d1; ++i)
        A[i] = new double[d2];  // allocate inner-most arrays (arrays of double, representing
rows)
    // Assign elements of vector x:
    for (int i = 0; i < x.Length; ++i)  // note use of the Length property
        x[i] = (double)i;
    // Assign elements of matrix A:
    for (int i = 0; i < A.Length; ++i)
        for (int j = 0; j < A[i].Length; ++j)  // note use of the Length property
            A[i][j] = (double)i + 0.1 * (double)j; // note double square brackets [][]
    // Perform multiplication of matrix A and vector x, store result in b:
    b = new double[A.Length];  // first, allocate the vector; use number of rows of A as its
dimension
    for (int i = 0; i < A.Length; ++i)
    {
        // Each elemennt of b will become dot product of the correspondig row of A and of x:
        double product = 0;  // auxiliary variable to calculate the dot product
        for (int j = 0; j < A[i].Length; ++j)
            product += A[i][j] * x[j];
        b[i] = product;
    }
    // Now, output elements of the matrix and vectors:
    Console.WriteLine("Matrix A (dimension " + A.Length + "*" + A[0].Length + "):");
    for (int i = 0; i < A.Length; ++i)
    {
        Console.Write("  ");
        for (int j = 0; j < A[i].Length; ++j)
            Console.Write(A[i][j] + "  ");
        Console.WriteLine();
    }
    Console.WriteLine();
    Console.WriteLine("Vector x (dimension " + x.Length + "):");
    for (int i = 0; i < x.Length; ++i)
        Console.WriteLine("  " + x[i] + "  ");
    Console.WriteLine();
    Console.WriteLine("Vector b = A*x (dimension " + b.Length + "):");
    for (int i = 0; i < b.Length; ++i)
        Console.WriteLine("  " + b[i] + "  ");
    Console.WriteLine(Environment.NewLine + "Matrix - vector multiplication example finished." +
Environment.NewLine);
```

11 NALOGA 3 – NUMERIČNO ODVAJANJE, INTEGRIRANJE IN DIFERENCIALNE ENAČBE

Ta naloga je sestavljena na podoben način kot naloge, ki jih bodo dobili študentje, ki v predvidenem roku pred vajami ne bodo uspeli oddati domače naloge s prejšnjih vaj. Namen takšnih

nlog je pridobivanje dodatnih praktišnih izkušenj, ki bodo v pomoč pri reševanju podobnih nalog, kot so bile podane pri domačih nalogah. Poleg same naloge je orisan tudi način reševanja.

V tej nalogi izvedemo odvajanje, integracijo in reševanje diferencialne enačbe na danem intervalu.

Rešujemo diferencialni enačbi z naslednjim začetnim pogojem na danem intervalu:

$$\begin{aligned} &g'(x) = g(x) + \cos(x) + \sin(x), \\ &x \in [0, 2\pi] \\ &g[0] = 0 \end{aligned} \quad .$$

ter

$$\begin{aligned} &f'(x) = \sin(x) + e^x, \\ &x \in [0, 2\pi] \\ &f[0] = 0 \end{aligned} \quad .$$

Diferencialni enačbi rešujemo po Eulerjevi metodi. Interval, na katerem rešujemo enačbi, razdelimo na *n* enako dolgih podintervalov dolžine *h*. Po vrsti izračunamo vrednosti funkcije v krajiščnih točkah podintervalov po formuli

$$f(x_i) = f[x_{i-1}] + h\, f'[x_{i-1}], i = 1, 2, ..., n \quad ,$$

pri čemer je začetna vrednost funkcije $f(x_0)$ podana. V vsaki iteraciji postopka izračunamo $f'(x_{i-1})$ iz vrednosti $f(x_{i-1})$, ki smo jo izračunali v prejšnji iteraciji, in iz diferencialne enačbe.

Numerična rešitev diferencialne enačbe bo predstavljena s tabelo *n*+1 vrednosti funkcije v vozliščih, ki delijo interval reševanja diferencialne enačbe na *n* podintervalov. Pri reševanju diferencialne enačbe poleg vrednosti funkcije v vozliščih neposredno dobimo tudi približke odvodov. To lahko uporabimo za preizkus, ali smo diferencialno enačbo pravilno reševali. Preizkus naredimo tako, da iz izračunanih vrednosti funkcije izračunamo numerične odvode in jih primerjamo z odvodi, ki jih izrčunamo iz predpisa diferencialne enačbe. Numerične odvode izračunamo po formuli

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} = \frac{f(x_{i+1}) - f(x_i)}{h}, i = 0, ..., n-1 \quad .$$

Odvod v točki x_n postavimo na isto vrednost, kot je izračunan odvod v x_{n-1}.

Iz predpisa diferencialne enačbe izračunamo npr. odvode za 1. funkcijo na nasleden način:

$$g'(x_i) = g(x_i) + \cos(x_i) + \sin(x_i) \quad .$$

Vidimo, da pri računanju odvodov po predpisu uporabimo samo vrednosti v točki, v kateri računamo odvod. Za preizkus, ali smo metodo pravilno implementirali, lahko preverimo ujemanje odvodov izračunanih po obeh metodah.

Poleg navedenega lahko izračunamo še določeni integral odvoda funkcije (ki je stranski produkt pri reševanju diferencialne enačbe) in ga primerjamo z rešitvijo diferencialne enačbe. Integral odvoda želimo primerjati z rešitvijo enačbe v vseh vozliščnih točkah delitve, ne le v zgornji meji integracijskega intervala. Rabili bomo torej metodo za numerično integriranje, ki izračuna integral v vseh točkah. Za to lahko uporabimo običajno metodo, ki jo pokličemo za vsak interval $[x_{i-1}, x_i], i = 1, ..., n$ posebej. Tak način pa je zelo neučinkovit, zato raje sprogramiramo metodo, ki sproti akumulira vrednost integrala v vozliščih po vrstnem redu. Uporabili bomo trapezno pravilo, pri katerem izračunamo integral F funnkcije f v točki x_i ,

$$F(x_i) = \int_{x_0}^{x_i} f(x)\, dx \ ,$$

po naslednjem iteracijskem predpisu:

$$F(x_0) = 0$$
$$F(x_i) = F(x_{i-1}) + \frac{h\left(f(x_{i-i}) + f(x_i)\right)}{2} \ .$$

Enostavno lahko preverimo, da imata obe diferencialni enačbi, ki ju rešujemo, isto analitično rešitev

$$f(x) = e^x - \cos(x) \quad .$$

Analitični odvod rešitve je

$$f'(x) = e^x + \sin(x) \quad .$$

To bomo uporabili pri primerjavi napak metod.

11.1 Naloga

V okviru te naloge naredi naslednje:

- Vsako posebej reši obe diferencialni enačbi, pri tem tabeliraj izračunane verdnosti **in** odvode.
- Tabeliraj analitično rešitev $f(x)$ in njen odvod $f'(x)$

- Izračunaj numerični odvod analitične funkcije $f(x)$ in numerični integral njenega analitičnega odvoda $f'(x)$.
- Izračunaj numerični integral tabeliranega odvoda funkcije, ki ga dobiš pri reševanju diferencialne enačbe, ter numerični odvod tabelirane rešitve diferencialne enačbe za obe funkciji.

Za vrednosti *n*=30, *n*=100, *n*=200 in *n*=1000 primerjaj:

- Analitično funkcijo $f(x)$ z numeričnim integralom $f'(x)$
- Analitično funkcijo $f'(x)$ z numeričnim odvodom $f(x)$
- Reštev diferencialnih enačb z analitično funkcijo $f(x)$
- Odvod funkcije, ki ga dobiš pri reševanju diferencialnih enačb, z analitičnim odvodom $f'(x)$
- Numerični odvod funkcije, ki jo dobiš pri reševanju diferencialnih enačb, z odvodom, ki ga dobiš pri reševanju diferencialnih enačb.
- Numerični integral odvoda, ki ga dobiš pri reševanju diferencialnih enačb, z numerično rešitvijo diferencialne enačbe
- Numerični odvod funkcije, ki jo dobiš pri reševanju diferencialnih enačb, z analitičnim odvodom $f'(x)$

Pri *n*=30 primerjaj vrednosti v vseh točkah. Pri ostalih izračunaj samo mero za napako, ki je definirana kot

$$\varepsilon_r = \max_i \left(\left| \tilde{f}(x_i) - f(x_i) \right| \right) / \max_i \left| f(x) \right|, i = 0, ..., n \ .$$

Tu je $\tilde{f}(x_i)$ približna vrednost funkcje $f(x)$ v točki x_i.

11.2 Reševanje

Pri reševanju takšnih nalog si najprej pripravimo načrt reševanja. Najprej identificiarmo, kakšne metode bomo rabili za rešitev problema, od najbolj osnovnih do tistih, ki so odvisne od le-teh. Nato te osnovne gradnike povežemo v višjenivojske in rešimo roblem po delih.

Za rešitev pričujočega primera bomo rabili naslednje osnovne metode::

- metoda za **vzorčenje**, ki naredi tabelo *n*+1 točk x_i
- metoda za tabeliranje brednosti funkcije, ki iz zabele x_i naredi tabelo vrednosti $f(x_i)$. Argumet metode bo delegat, ki predstavlja realno funkcijo ene spremenljivke.
- Metoda za integriranje tabele tako, da so v tabeli rezultatov shranjeni vsi novi rezultati

- Metoda za numerično odvajanje na podoben način
- Metoda za numerično intgiranje na podoben način.
- Metoda za reševanje diferencialne enačbe. Vhodna podatka sta tabela abscis in delegat, ki ima dva realna argumenta (absciso točke in vrednost funkcije v tej točko)
- Metoda za primerjavo dveh tabel
- Mnetoda za izračun napake iz dveh tabel

Metode za numerično integriranje, odvajanje in reševanje diferencialnih enačb sprogramiramo tako, da kot vhodni podatek vzamejo tabele vrednosti in ne funkcijske predpise.

12NALOGA 4 – REŠEVANJE SISTEMOV LINEARNIH ENAČB

V tej nalogi rešujemo sisteme linearnih enačb oblike

$$\mathbf{Ax} = \mathbf{b},$$

kjer je **A** matrika sistema z *n* vrsticami in *n* stolpci, **b** vektor desnih strani dolžine *n* in **x** vektor rešitev, ki zadošča zgornji enačbi.

Sisteme bomo reševali z uporabo numerične knjižnice *Math.Net Iridium*. Demonstracija uporabe knjižnice za množenje matrik in reševanje sistemov enačb je na FTP v direktoriju *csharp\gresovnik\05LinearAlgebra* .

12.1 Naloga

Reši sisteme linearnih enačb z *n* neznankami, kjer je $n \in \{2, 4, 8, 16, 32, ...\}$. Matriko sistema **A** in vektor desnih strani **b** generiraj naključno z uporabo metode *Matrix.Random()*. Vsak sistem reši najprej z uporabo razcepa **LU**, nato pa *isti sistem* še z uporabo razcepa **QR**. Za vsak *n* primerjaj normo ostanka

$$r = \|\mathbf{Ax} - \mathbf{b}\|,$$

Število neznank povečuj za faktor 2, dokler skupni čas rešvevanja sistema enačb po obeh metodah ne postane večji od minute. Pri tem lahko opazuješ, kako je do neke meje čas računanja komaj opazen, nato pa zelo hitro narašča s številom neznank. To je zaradi tega, ker je vodilni člen pri številu operacij za obe metodi sorazmeren z n^3.

Naredi tabelo, s katero primerjaš norme ostankov pri obeh metodi za vse *n*, pri katerih rešuješ sistem enačb. Pri katerem *n* (približno) čas računana preseže minuto?

Ko bo čas presegel minuto, izvedi za občutek reševanje sistema enačb še za nekaj naslednjih *n*, ki ustrezajo potencam števila 2.

12.2Namig

Pri delo s knjižnico *Math.Net Iridium* za predstavitev desnih strani in rešitve sistema ne uporabiš tipa *Vector*, ampak kar tip *Matrix* dimenzije *n**1. Za normo uporabiš Forbeniusovo normo, ki pri matrikah z enim stolpcem ustreza evklidski normi ustreznega vektorja.

13NALOGA 5A – ROBNI PROBLEM

Z metodo po svoji izbiri (ali s streljanjem ali z relaksacijo) reši robni problem

$$y''-y=-2e^{x}\ ,\quad y(0)=1,\ y(1)=0\ .$$

Analitična rešitev je

$$y=(1-x)e^{x}\ .$$

Problem reši pri različnih delitvah intervala: n=50 in n = 200.

Opise metod najdeš med drugim na naslednjih straneh:

- http://homepage.univie.ac.at/franz.vesely/cp_tut/nol2h/new/c4od_s3bv.html
- http://www.aip.de/groups/soe/local/numres/bookcpdf/c17-1.pdf (streljanje)
- http://www.aip.de/groups/soe/local/numres/bookcpdf/c17-3.pdf (relaksacija)

14NALOGA 5B – MINIMIZACIJA FUNKCIJ

Z metodo iz knjižnice najdi minimum funkcije

$$f(x,y)=(1-x)^{2}+100\left(y-x^{2}\right)^{2}\ .$$

Pri vsakem približku, pri katerem metoda izračuna vrednost funkcije, izpiši vrednosti neodvisnih spremenljivk, vrednost funkcije in oddaljenost od minimuma. Minimum funkcije je v (1,1).

Za minimizacijo uporabi metdo L-BFGS iz knjižnice alglib.net. Spletna stran knjižnice je na naslovu http://www.alglib.net/ . Na spletni strani dobiš izvorno kodo metode, ki jo vključiš v svoj projekt. Spletna stran vsebuje še razlago algoritma in navodila za uporabo metode s primeri. Potrebne datoteke v C# s spletne strani vključiš v svoj projekt.

Koda metode L-BFGS in njena spletna stran z navodili sta vključeni tudi rešitev test-solution v projektu AlgLib, ki se nahaja v direktoriju csharp/external/ALGLIB/. Navodila so v datoteki doc/lbfgs.php.htm, odprete jih lahko tudi neposredno iz Visual Studia (z desnim gumbom kliknete na datoteko in izberete ”View in Browser”). Metodo L-BFGS lahko uporabite enostavno tako, da v svojo rešitev vključite projekt AlgLib in se sklicujete nanj. V projekt je vključena starejša verzija kode in je tam le informativno ter za primer, da bi bila spletna stran projekta AlgLib nedostopna.