Megjegyzések:

Ez a dokumentum a 2017-es tavaszi fiznum2 gyakorlat házi feladatait, és annak általam beadott megoldásait tartalmazza. Összesen 150 pontot kaptam rájuk, a vizsgázáshoz 120-ra volt szükség.
Amennyiben a további években is ugyanezek a feladatok szükségesek a tárgy teljesítéséhez, senkinek nem ajánlom, hogy ezeket a kódokat adja be, mivel nagyon egyszerűen kideríthető a másolás. Csupán azért töltöttem fel, hogy segítséget nyújthasson a “mire gondolt a költő, amikor kiírta a feladatokat” mondatra.
A feltöltés idejében a kódok gond nélkül fordultak, remélem segítenek a sajátod megalkotásában, ha elakadtál. Elírások természetesen előfordulhatnak.

1. feladat: Egyszerű mátrixműveletek
Minimum feladat: Írjunk olyan programot, amely parancssorban megadott MxN méretű mátrixot (pl. matrix.dat) és egy N elemű oszlopvektort (pl. vector.dat) fájlból beolvas, összeszorozza őket, és az eredményvektort kiírja a standard outputra! A parancssor M=5, N=3 esetén így nézhet ki:
matmul 5 3 matrix.dat vector.dat
Teljes feladat: Írjuk meg a mátrixot fájlból betöltő függvényt úgy, hogy az ne várja előre a mátrix méretét, hanem találja ki a fájlban levő oszlopok és sorok számából. A fájl formátumára nincsen sok megkötés, a lényeg, hogy ránézésre mátrixot tartalmazó szöveges állomány legyen. Készítsünk a programhoz különböző próbafájlokat, amikkel a helyes működése tesztelhető.
Figyelem: a fájlt csak egyszer és csak szekvenciálisan szabad végigolvasni, valamint a fájl tetszőlegesen nagy lehet!

2. feladat: Gauss-elimináció
Minimum feladat: Írjunk olyan programot, amelyik beolvas egy NxN méretű valós mátrixot (szöveges, szóközzel, illetve soremeléssel elválasztott fájlokból), eredményként pedig kiírja a mátrix inverzét! Algoritmusként használjuk a Gauss- eliminációt sor- és oszlopcserékkel! A program detektálja, ha az invertálandó mátrix szinguláris! Tervezzük meg modulárisan a programot, úgy, hogy egy lineárisegyenlet- megoldó algoritmus függvényként legyen hívható!
Teljes feladat: A Gauss-elimináció implementálása után használjuk mátrix inverzének meghatározására a LAPACK csomag szinguláris érték-dekompozíció függvényét. Hasonlítsuk össze az SVD algoritmus eredményét a Gauss- eliminációéval néhány mátrix esetében! Teszteljük a két algoritmus sebességét különböző méretű mátrixokra! Próbáljuk ki az SVD-t szinguláris és közel szinguláris mátrixokra.

3. feladat: Polinomillesztés
Minimum feladat: Írjunk olyan programot, amely az előző feladatban kidolgozott, lineáris egyenletrendszereket megoldó kódot felhasználva általános N változós polinomillesztést hajt végre a lineáris legkisebb négyzetek módszerével! A program parancssori paraméterként várja a független változók számát, az illesztendő polinom rendjét, valamint a bemenő adatokat tartalmazó fájl nevét. A program írja ki a képernyőre az illesztési paramétereket, valamint egy fájlba az eredeti mérési értékeket és az illesztett polinom által adott becsléseket az eredeti bemeneti fájl minden adatpontjára. Készítsünk ábrát, melyről leolvasható a polinomillesztés jósága! Készítsünk saját adatfájlokat az algoritmus tesztelésére, majd futassuk a programot az alább letölthető adatfájlokra

(5 változó). Az alapfeladathoz nem szükséges a többváltozós polinomok vegyes tagjait (pl. xy, xy^2 stb.) figyelembe venni. A bemenő adatokat tartalmazó fájl formátuma: Szöveges fájl, mely soronként N + 2 számot tartalmaz, ahol N a független változók száma. Az első N szám ennek megfelelően a független változók értéke, az (N+1). a függő változó mért értéke, az (N+2). szám pedig a mérés abszolút hibája.

A kimenő fájl formátuma: Szöveges fájl, mely soronként két számot tartalmaz: a mérési értéket (az eredeti fájl (N+1). oszlopából), illetve a becsült értéket (az eredeti fájl első N oszlopát a polinomba helyettesítve). A kimeneti fájl annyi sorból álljon, mint a bemeneti fájl.
Nem kötelező feladat (jegybe nem számít bele): Valósítsunk meg nem lineáris függvényillesztést MCMC módszerrel, Metropolis-Hastings-algoritmussal. A program teszteléséhez készítsünk saját adatfájlokat. Próbálkozzunk Gauss- illesztéssel!

4. feladat: Differenciálegyenlet megoldása
Minimum feladat: Írjunk olyan programot, amelyik az explicit Euler-módszer segítségével megoldja a Föld- Hold- rendszer mozgásegyenletét. Ábrázoljuk a kapott eredményeket. Nézzük meg, hogy tapasztalható-e eltérés az analitikus megoldástól sok periódus után! A feladat megoldása során dolgozzunk síkban, geocentrikus koordinátákban (a Föld a 0,0 pontban van rögzítve). Ábrázoljuk a Hold pályáját, valamint a rendszer teljes energiáját az idő függvényében.
Föld tömege: 5.9736×10^{24}kg
Hold tömege: 7.349×10^{22}kg
Apogeum távolsága: 405.500 km
Sebesség apogeumban: 964 m/s
Perigeum távolsága: 363.300 km
Sebesség perigeumban: 1076 m/s
Gravitációs állandó: $6.67384 \times 10^{-11}m^3kg^{-1}s^{-2}$
Teljes feladat: Integráljuk a mozgásegyenletet egyszerű, valamint adaptív lépéshossz-szabályozott negyedrendű Runge-Kutta-módszerrel! A program legyen teljesen általános, ne legyen megkötés az integrálandó változók számára. Használjunk függvénypointereket! Próbáljuk ki a programot a Lorenz-féle egyenletrendszer, vagy a kettős inga egyenleteinek megoldására! Ábrázoljuk az eredményeket!

1. feladat:
(50/50 pont, teljes megoldás)

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
double* matrixbe(char** argv, int* msize)
{
    char c;
    int temp = 1;
    int max = 0;
    double* matrix = (double*)malloc(sizeof(double));
    FILE* matrixFile = fopen(argv[1], "r");
    while (fscanf(matrixFile, "%lf%c", &matrix[*msize], &c) > 0)
    {
        (*msize)++;
        if (max != 0 && (*msize) > max)
        {
            fclose(matrixFile);
            free(matrix);
            return 0;
        }
        if (c == '\n' && temp == 1)
        {
            matrix = (double*)realloc(matrix, (*msize) * (*msize) * sizeof(double));
            temp = 0;
            max = (*msize) * (*msize);
        }
        if (c == ' ' && temp == 1)
        {
            matrix = (double*)realloc(matrix, ((*msize) + 1) * sizeof(double));
        }
    }
    *msize = (int)sqrt(*msize);
    fclose(matrixFile);
    return matrix;
}
double* vectorbe(char** argv, int* vsize)
{
    char c;
    double* vector = (double*)malloc(sizeof(double));
    FILE* vectorFile = fopen(argv[2], "r");
    while (fscanf(vectorFile, "%lf%c", &vector[*vsize], &c) > 0)
    {
        (*vsize)++;
        if (c == '\n')
        {
            vector = (double*)realloc(vector, ((*vsize) + 1) * sizeof(double));
        }
    }
    fclose(vectorFile);
    return vector;
```

```
}
double* multiplication(double* matrix, double* vector, int vsize)
{
    double* finalvector = (double*)malloc(vsize * sizeof(double));
    for (int i = 0; i < vsize; i++)
    {
        double temp = 0;
        for (int j = 0; j < vsize; j++)
        {
            temp += matrix[i * vsize + j] * vector[j];
        }
        finalvector[i] = temp;
    }
    return finalvector;
}
void vectorprint(double* vector, int size)
{
    for (int i = 0; i < size; i++)
    {
        printf("%lf\n", vector[i]);
    }
}
int main(int argc, char** argv)
{
    int msize = 0;
    int vsize = 0;
    double* matrix = matrixbe(argv, &msize);
    if (matrix == 0)
    {
        printf("Mátrix nem kvadratikus!");
        return 1;
    }
    double* vector = vectorbe(argv, &vsize);
    if (msize != vsize || msize == 0 || vsize == 0)
    {
        printf("A tagok méreteik miatt nem szorozhatóak össze.");
        return 1;
    }
    double* finalvector = multiplication(matrix, vector, vsize);
    vectorprint(finalvector, vsize);
    free(matrix);
    free(vector);
    free(finalvector);
    return 0;
}
```

2. feladat:
(40/70 pont, alap feladat megoldása)

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdbool.h>
double* Abe(char** argv, int* size)
{
    int tempsize = 0;
    char c;
    bool temp = true;
    double* A = (double*)malloc(sizeof(double));
    FILE* matrixFile = fopen(argv[1], "r");
    while (fscanf(matrixFile, "%lf%c", &A[tempsize], &c) > 0)
    {
        (tempsize)++;
        if (c == '\n' && temp)
        {
            A = (double*)realloc(A, (tempsize) * (tempsize) * sizeof(double));
            temp = false;
        }
        if (temp)
        {
            A = (double*)realloc(A, ((tempsize) + 1) * sizeof(double));
        }
    }
    fclose(matrixFile);
    *size = (int)sqrt(tempsize);
    return A;
}
double* Ibe(int size)
{
    double* I = (double*)malloc(size * size * sizeof(double));
    for (int i = 0; i < size * size; i++)
    {
        I[i] = 0;
    }
    for (int i = 0; i < size * size; i += size + 1)
    {
        I[i] = 1;
    }
    return I;
}
double* Vbe(char** argv, int size)
{
    double* V = (double*)malloc(size * sizeof(double));
    FILE* vectorFile = fopen(argv[2], "r");
    for (int i = 0; i < size; i++)
    {
        fscanf(vectorFile, "%lf ", &V[i]);
    }
```

```
    fclose(vectorFile);
    return V;
}
void matrixprint(double* matrix, int size, char* nev)
{
    printf("%s\n", nev);
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            printf("%lf ", matrix[i * size + j]);
        }
        printf("\n");
    }
    printf("\n");
}
void vectorprint(double* vector, int size, char* nev)
{
    printf("%s\n", nev);
    for (int i = 0; i < size; i++)
    {
        printf("x%d=%lf\n", i + 1, vector[i]);
    }
    printf("\n");
}
void gaussJordanInvertalas(double* A, double* I, int size)
{
    int temp1;
    double temp2;
    double temp3;
    for (int j = 0; j < size; j++)
    {
        temp1 = j;
        for (int i = j + 1; i < size; i++)
        {
            if (A[i * size + j] > A[temp1 * size + j])
            {
                temp1 = i;
            }
        }
        if (fabs(A[temp1 * size + j]) < 0.0005)
        {
            printf("\nA matrix numerikusan szingularis.\n");
            exit(1);
        }
        if (temp1 != j)
        {
            for (int k = 0; k < size; k++)
            {
                temp2 = A[j * size + k];
                A[j * size + k] = A[temp1 * size + k];
                A[temp1 * size + k] = temp2;
```

```
            temp2 = I[j * size + k];
            I[j * size + k] = I[temp1 * size + k];
            I[temp1 * size + k] = temp2;
        }
    }
    for (int i = 0; i < size; i++)
    {
        if (i != j)
        {
            temp3 = A[i * size + j];
            for (int k = 0; k < size; k++)
            {
                A[i * size + k] -= (A[j * size + k] / A[j * size + j]) * temp3;
                I[i * size + k] -= (I[j * size + k] / A[j * size + j]) * temp3;
            }
        }
        else
        {
            temp3 = A[i * size + j];
            for (int k = 0; k < size; k++)
            {
                A[i * size + k] /= temp3;
                I[i * size + k] /= temp3;
            }
        }
    }
  }
}
void linEgyenlet(double* A, double* V, int size)
{
  double temp;
  for (int j = 0; j < size; j++)
  {
    for (int i = 0; i < size; i++)
    {
        if (i != j)
        {
            temp = A[i * size + j] / A[j * size + j];
            for (int k = 0; k < size; k++)
            {
                A[i * size + k] -= temp * A[j * size + k];
                V[i] -= temp / size * V[j];
            }
        }
    }
  }
  for (int i = 0; i < size; i++)
  {
    V[i] /= A[i * size + i];
    A[i * size + i] /= A[i * size + i];
  }
}
```

```
int main(int argc, char** argv)
{
    int size = 0;
    if (argc >= 3)
    {
        double* A = Abe(argv, &size);
        double* V = Vbe(argv, size);
        linEgyenlet(A, V, size);
        matrixprint(A, size, "A:");
        vectorprint(V, size, "Az egyenletrendszer megoldasa:");
        free(A);
        free(V);
        return 0;
    }
    else if (argc == 2)
    {
        double* A = Abe(argv, &size);
        double* I = Ibe(size);
        gaussJordanInvertalas(A, I, size);
        matrixprint(I, size, "A mátrix inverze:");
        free(A);
        free(I);
        return 0;
    }
    else
    {
        printf("Nincs eleg parameter!");
        return 0;
    }
}
```

3. feladat:
(30/50 pont, alap feladat megoldása)

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
double* beolvasXYH(char** argv, int* sor, int fuggetlen)
{
    double temp;
    int index = 0;
    char c;
    double* XYH = (double*)malloc((fuggetlen + 2) * sizeof(double));
    FILE* File = fopen(argv[3], "r");
    while (fscanf(File, "%lf%c", &temp, &c) > 0)
    {
        XYH[*sor * (fuggetlen + 2) + index] = temp;
        index++;
        if (index == fuggetlen + 2)
        {
            index = 0;
            (*sor)++;
            if (c == '\n')
            {
                XYH = (double*)realloc(XYH, ((*sor) + 1) * (fuggetlen + 2) * sizeof(double));
            }
        }
    }
    fclose(File);
    return XYH;
}
void szeparal(double* XYH, double* X, double* Y, int sor, int rend, int fuggetlen)
{
    for (int i = 0; i < sor; i++)
    {
        X[i * (fuggetlen * rend + 1)] = 1;
        for (int j = 0; j < fuggetlen; j++)
        {
            for (int k = 0; k < rend; k++)
            {
                X[i * (fuggetlen * rend + 1) + j * rend + k + 1]
                    = (pow(XYH[i * (fuggetlen + 2) + j], k + 1))
                    / XYH[i * (fuggetlen + 2) + fuggetlen];
            }
        }
        Y[i] = XYH[i * (fuggetlen + 2) + fuggetlen] / XYH[i * (fuggetlen + 2) + fuggetlen + 1];
    }
}
double* transponalt(double* X, int sor, int oszlop)
{
    double* XT = (double*)malloc(sor * oszlop * sizeof(double));
    for (int i = 0; i < sor; i++)
    {
```

```
        for (int j = 0; j < oszlop; j++)
        {
            XT[j * sor + i] = X[i * oszlop + j];
        }
    }
    return XT;
}
double* matrixSzorzas(double* XT, double* X, int sor, int oszlop)
{
    double sum = 0;
    double* XTX = (double*)malloc(oszlop * oszlop * sizeof(double));
    for (int i = 0; i < oszlop; i++)
    {
        for (int j = 0; j < oszlop; j++)
        {
            for (int k = 0; k < sor; k++)
            {
                sum += XT[i * sor + k] * X[k * oszlop + j];
            }
            XTX[i * oszlop + j] = sum;
            sum = 0;
        }
    }
    return XTX;
}
double* vectorSzorzas(double* XT, double* Y, int sor, int oszlop)
{
    double* XTY = (double*)malloc(sor * sizeof(double));
    for (int i = 0; i < sor * oszlop; i += sor)
    {
        double sum = 0;
        for (int j = 0; j < sor; j++)
        {
            sum += XT[i + j] * Y[j];
        }
        XTY[i / sor] = sum;
    }
    return XTY;
}
double* illesztes(double* X, double* Y, double* XYH, int sor, int oszlop, int fuggetlen)
{
    double* Yill = (double*)malloc(sor * sizeof(double));
    for (int i = 0; i < sor; i++)
    {
        double sum = 0;
        for (int j = 0; j < oszlop; j++)
        {
            sum += X[i * oszlop + j] * Y[j] * XYH[i * (fuggetlen + 2) + fuggetlen + 1];
        }
        Yill[i] = sum;
    }
    return Yill;
```

```
}
void linEgyenlet(double* XTX, double* XTY, int size)
{
    double temp;
    for (int j = 0; j < size; j++)
    {
        for (int i = 0; i < size; i++)
        {
            if (i != j)
            {
                temp = XTX[i * size + j] / XTX[j * size + j];
                for (int k = 0; k < size; k++)
                {
                    XTX[i * size + k] -= temp * XTX[j * size + k];
                    XTY[i] -= temp / size * XTY[j];
                }
            }
        }
    }
    for (int i = 0; i < size; i++)
    {
        XTY[i] /= XTX[i * size + i];
        XTX[i * size + i] /= XTX[i * size + i];
    }
}
void kiirPolinomEgyutthatok(double* XTY, int oszlop)
{
    FILE* File = fopen("PolinomEgyutthatok.dat", "w");
    for (int i = 0; i < oszlop; i++)
    {
        fprintf(File, "%lf\n", XTY[i]);
    }
    fclose(File);
}
void kiirPolinomFuggoErtekek(double* Yillesztett, double* XYH, int sor, int fuggetlen)
{
    FILE* File = fopen("PolinomFuggoErtekek.dat", "w");
    for (int i = 0; i < sor; i++)
    {
        fprintf(File, "%lf %lf\n", XYH[(i * (fuggetlen + 2)) + fuggetlen], Yillesztett[i]);
    }
    fclose(File);
}
int main(int argc, char** argv)
{
    int fuggetlen = atoi(argv[1]), rend = atoi(argv[2]), sor = 0, oszlop = fuggetlen * rend + 1;
    double* XYH = beolvasXYH(argv, &sor, fuggetlen);
    double* X = (double*)malloc(sor * (fuggetlen * rend + 1) * sizeof(double));
    double* Y = (double*)malloc(sor * sizeof(double));
    szeparal(XYH, X, Y, sor, rend, fuggetlen);
    double* XT = transponalt(X, sor, oszlop);
    double* XTX = matrixSzorzas(XT, X, sor, oszlop);
```

```
    double* XTY = vectorSzorzas(XT, Y, sor, oszlop);
    linEgyenlet(XTX, XTY, oszlop);
    double* Yillesztett = illesztes(X, XTY, XYH, sor, oszlop, fuggetlen);
    kiirPolinomEgyutthatok(XTY, oszlop);
    kiirPolinomFuggoErtekek(Yillesztett, XYH, sor, fuggetlen);
    free(XT);
    free(Y);
    free(Yillesztett);
    free(XYH);
    free(XTY);
    free(X);
    free(XTX);
    return 0;
}
```

4. feladat:
(30/60 pont, alap feladat megoldása)

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
double X(double x, double y, double Fold, double Hold, double G)
{
   return -G * Fold * Hold * x / pow(x * x + y * y, 1.5);
}

double Y(double x, double y, double Fold, double Hold, double G)
{
   return -G * Fold * Hold * y / pow(x * x + y * y, 1.5);
}
void Euler(double* DX, double* DY, double dt, double Fold, double Hold, double G,
double Apogtav,
   double Apogseb, double N)
{
   double x = 0, y = Apogtav, vx = Apogseb, vy = 0, x_n1, y_n1, vx_n1, vy_n1, XX, YY;
   int i = 0;
   while (i < N)
   {
      XX = X(x, y, Fold, Hold, G);
      YY = Y(x, y, Fold, Hold, G);
      x_n1 = x + vx * dt;
      vx_n1 = vx + XX * dt / Hold;
      y_n1 = y + vy * dt;
      vy_n1 = vy + YY * dt / Hold;
      x = x_n1;
      vx = vx_n1;
      y = y_n1;
      vy = vy_n1;
      DX[i] = x;
      DY[i] = y;
      i++;
   }
}
void KiirFileba(double* DX, double* DY, int N)
{
   FILE* File = fopen("megold.dat", "w");
   int i = 0;
   while (i < N)
   {
      fprintf(File, "%lf%lf\n", DX[i], DY[i]);
      i++;
   }
   fclose(File);
}
int main()
{
   const double Fold = 5.9736e24, Hold = 7.349e22, ApogTav = 4.055e8, ApogSeb = 965,
```

```
            PerigTav = 3.633e8, PerigSeb = 1.076e3, G = 6.67384e-11;
    int N;
    double dt;
    printf("Add meg dt-t, és N-t (a pontok szamat) !");
    printf("\nAdd meg 'dt'-t:");
    scanf("%lf", &dt);
    printf("\nAdd meg 'N'-t:");
    scanf("%d", &N);
    double* DX = (double*)malloc(N * sizeof(double));
    double* DY = (double*)malloc(N * sizeof(double));
    Euler(DX, DY, dt, Fold, Hold, G, ApogTav, ApogSeb, N);
    KiirFileba(DX, DY, N);
    return 0;
}
```