

Eötvös Loránd Tudományegyetem

Informatikai Kar

Programozási Nyelvek és Fordítóprog-  
ramok Tanszék

---

# Interpoláció osztott rendszereken

Tejfel Máté  
egyetemi tanár

Cselyuszká Alexandra  
Informatika Bsc

Budapest, 2015

# Tartalomjegyzék

<b>1. Bevezetés</b>	<b>2</b>
1.1. Feladat elemzése . . . . .	2
1.2. Feladat megvalósítása . . . . .	3
<b>2. Felhasználói dokumentáció</b>	<b>4</b>
2.1. Bevezetés . . . . .	4
2.2. Telepítési útmutató . . . . .	4
2.2.1. Rendszer követelmények . . . . .	4
2.2.2. Segédprogramok telepítése . . . . .	4
2.2.3. Szerver és segédgépek üzembe helyezése . . . . .	4
2.2.4. Használati útmutató . . . . .	4
<b>3. Fejlesztői dokumentáció</b>	<b>5</b>
3.1. Megoldási terv . . . . .	5
3.2. Weboldal . . . . .	6
3.2.1. Felépítés . . . . .	6
3.2.2. Fontosabb objektumok és függvények . . . . .	7
3.2.3. Kommunikáció . . . . .	11
3.2.4. Tesztelési terv . . . . .	11
3.3. Elosztott rendszer . . . . .	11
3.3.1. Web-szerver kommunikáció . . . . .	11
3.3.2. Gép-szerver kommunikáció . . . . .	11
3.3.3. Elosztás folyamata . . . . .	11
3.3.4. Tesztelési terv . . . . .	11
3.4. Kalkulátor . . . . .	11
3.4.1. Fontosabb számítási függvények . . . . .	11
3.4.2. Elosztott rendszerrel való kommunikáció . . . . .	11
3.4.3. Tesztelési terv . . . . .	12

# 1. fejezet

## Bevezetés

*"A gyakorlatban sokszor felmerül olyan probléma, hogy egy nagyon költségesen kiszámítható függvénnyel kellene egy megadott intervallumon dolgoznunk. Ekkor például azt tehetjük, hogy néhány pontban kiszámítjuk a függvény értékét, majd keresünk olyan egyszerűbben számítható függvényt, amelyik illeszkedik az adott pontokra." [1]*

A szakdolgozatom célja ezekre a problémákra megoldást adni elosztott környezetben.

### 1.1. Feladat elemzése

Adott pontthalmazokból kívánunk egy közelítő polinómot becsülni. Ezeket különböző interpolációs technikával meg tudjuk adni, ki tudjuk számolni. Több interpolációs technika létezik, melyekből könnyen meg tudunk adni akár több polinómot is egy adott pontthalmazhoz.

Ezekkel a számításokkal előfordulhat, hogy lassan futnak, főleg ha több interpolációt kívánunk egyszerre számolni. Ebben az esetben optimálisabb több gépen számolni a különböző pontthalmazokat.

Ebben a feladatban egy speciális megvalósítása lesz ennek a számításnak.

A grafikus része egy weboldal, melyen szerkeszthetjük a pontthalmazokat. A számítás részét egy szerver végzi amely figyeli a felcsatlakozó gépeket. Amikor kap egy számítandó adathalmazt, akkor több gép segítségével kiszámítja az eredményt. Ha minden részfeladat végzett, akkor vissza küldi a weboldalra, ahol az eredmények megtekinthetők grafikus formában.

## 1.2. Feladat megvalósítása

A **grafikus felület** egy weboldal, mely javascript-ben és HTML-ben van megvalósítva. A felületen egy listát tekinthetünk meg, ahova több ponthalmazt is felvehetünk. Mentés hatására az értékek a háttérben eltárolódnak. A ponthalmazok közül választhatunk egyet, amely betölődik felületre.

A szerkesztő felület egy táblázatból és egy grafikonból áll, emellett még a különböző speciális számításra vonatkozó tulajdonságok (interpoláció típusa) valamint a grafikonon való megjelenítéshez tartozó tulajdonságok (polinóm pontosság, megtekintendő intervallum) is szerkeszthetők.

Ha befejeztük a halmazok szerkesztését elküldhetjük a számítani kívánt értékeket a szerver felé.

A **szerver** feladata hogy figyelje a felületről érkező adatokat. Ha az adathalmaz megérkezett, akkor a szerver kibontja az adatokat egy JSON-ból, és elindítja az elosztást.

Az elosztáshoz a szerveren el kell indítani egy figyelő folyamatot amelyre lehetősége van egy külső gépnek felcsatlakozni. Amikor a szerveren indul egy számolás a felcsatlakozott gépeket lekérdezi, majd a feladatokat szétosztja.

A szerver megvalósítása és a gépekre való szétosztás Erlang-ban lett megvalósítva. A JSON feldolgozásához Mochi-JSON lett alkalmazva. A feldolgozás után az adathalmazon végig megyünk és azok alapján felparaméterezzük, és meghívjuk a számítást végző függvényt.

A számításhoz használt maximális gépek száma paraméterként megadható, de a tényleges számítást csak annyi gépen tudjuk maximálisan végezni ahány gép felcsatlakozott a számításhoz.

A **számítás** megvalósítása C++ nyelven történt. A paraméterek alapján a Lagrange-féle, Newton-féle, Hermite-féle interpolációs technikák közül eldönti melyik esetet használja.

Valamint inverz interpolációt is választhatnak a Lagrange vagy a Newton interpoláció esetén.

A programban implementálásra került egy egyszerű polinóm szorzás és összeadás, valamint az interpolációkhoz szükséges függvények. Lagrange számítás a polinóm műveletek és a képlet felhasználásával valósult meg. Newton és Hermite esetén a kapott adatokból először a kezdő mátrixot kell legenerálni, majd kiszámítani.

Abban az esetben ha Newton vagy Lagrange polinómot számolunk nem vesszük figyelembe a derivált pontokat, viszont figyelembe vesszük ha inverz számítást kívánunk végezni.

## 2. fejezet

# Felhasználói dokumentáció

### 2.1. Bevezetés

### 2.2. Telepítési útmutató

#### 2.2.1. Rendszer követelmények

#### 2.2.2. Segédprogramok telepítése

#### 2.2.3. Szerver és segédgépek üzembe helyezése

#### 2.2.4. Használati útmutató

Weboldal

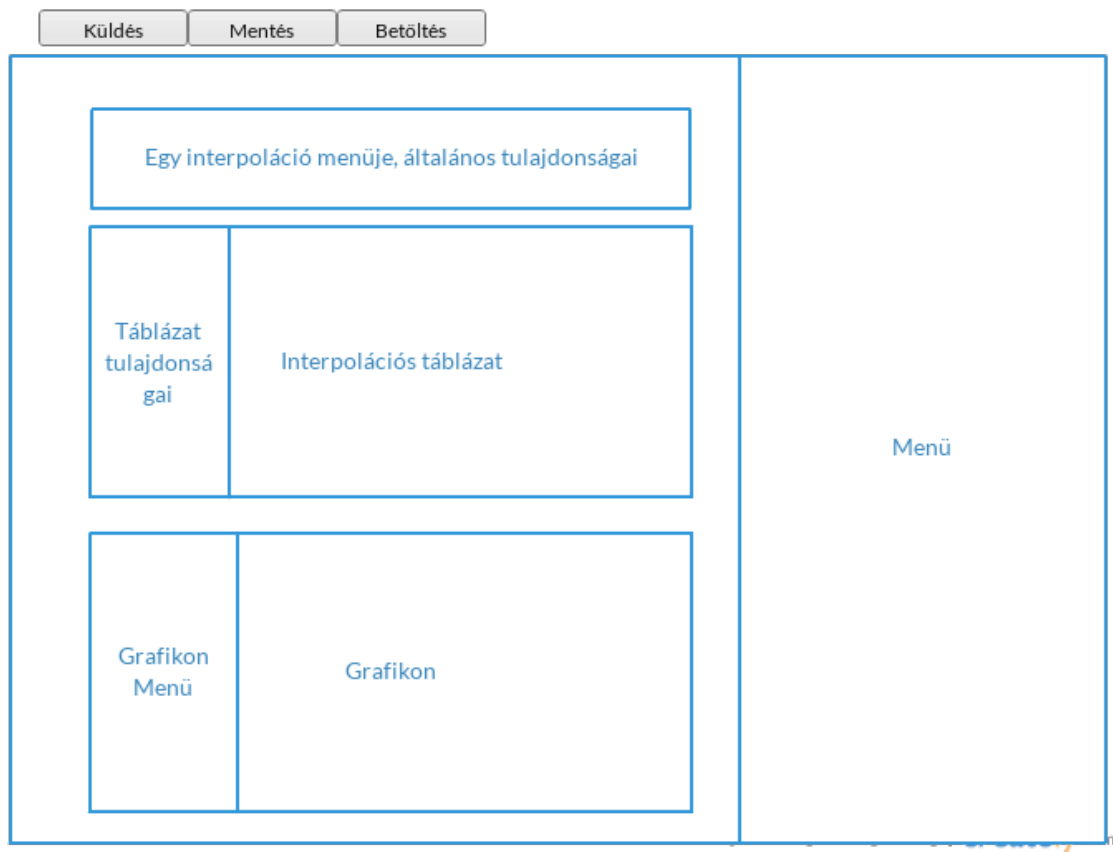
Szerver

## 3. fejezet

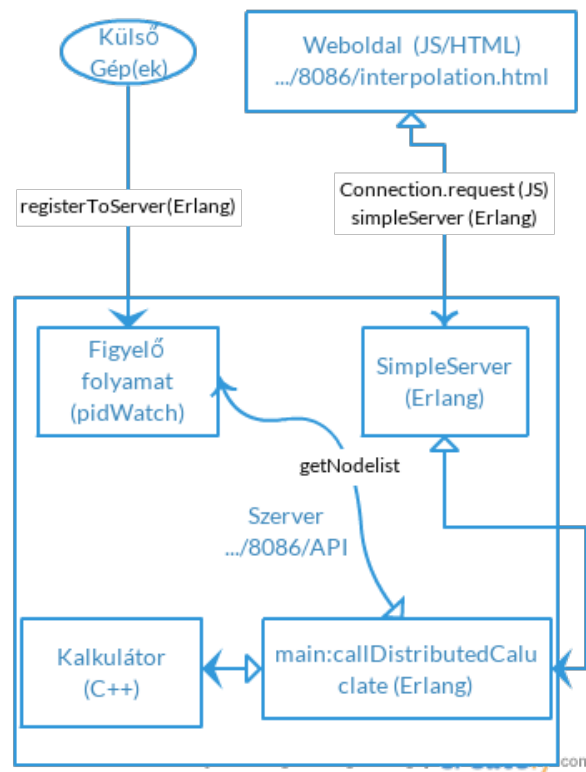
# Fejlesztői dokumentáció

### 3.1. Megoldási terv

A programot 3 nagy részre lehet bontani:



3.1. ábra. Weboldal Vázlata



3.2. ábra. Kommunikáció

## 3.2. Weboldal

Weboldal felépítése HTML és JavaScript segítségével valósult meg. Egy oldalból áll melyen a felhasználó össze állítja a neki szükséges adathalmazt. Új adathalmazokat hozhat létre, a régieket szerkesztheti. A háttérben JSON-be formálódnak az adatok. Ha a felhasználó végzett egy gombra nyomással a program legenerálja a szükséges objektumot.

A felület sok gombot tartalmaz, melyek hatására frissíthetők az adatok. Amikor frissítünk egy részt, általában mentődnek az értékek egy inputba JSON formában.

### 3.2.1. Felépítés

A weboldal forráskódja a Webpage mappában helyezkedik el. A fájlszerkezet az alábbi:

#### webpage.js :

Globális változók inicializálása és pár alapbeállítás lefuttatása

#### webpage.html :

Weboldal megjelenítése, fájlok betöltése

#### init

Inicializáló függvények hívásai és események

**menulist.js** : Interpolációk listájának inicializálója

**plot.js** : Interpolációs grafikon inicializálása

**table.js** : Interpolációs táblázat inicializálása

**events.js** : Gombra kattintások eseményei

### model

Objektumok, melyeket az inicializáló lépésben hívunk, és azok segédletei

**base.js** : Globális függvények

Base.get, Base.erlangJSON, Base.forEach

**base\_table.js** : Általános táblázat generáló függvény

**connection.js** : Szerver kapcsolat meghívására szolgáló függvény

Connection.request

**plot\_types.js** : A grafikon kirajzoló típus objektumai

**polinome.js** : Polinóm kirajzolását segítő függvények

makePolinome található benne és egyéb segédfüggvények

**web\_page\_debug.js** : A Weboldalon történő logolást segítő objektum

Jelenleg sehol nem használjuk már, de a megvalósítás során fontos szerepe volt a hibajavításban

### model/interpolation

Az oldal 3 fő részegységének függvényei

**menulist.js** : Interpolációk lista megvalósítása

function interpolationMenulist (aConfig) Objektum fájlja

**plot.js** : Interpolációs grafikon megvalósítása

function interpolationPlot(aConfig) Objektum fájlja

**table.js** : Interpolációs táblázat megvalósítása

function interpolationTable(aConfig) Objektum fájlja

### 3.2.2. Fontsabb objektumok és függvények

Az objektumokat legtöbb esetben egy függvény generálja, melyben that.-al jelöltek azok, melyeket a visszatérés után felhasználunk az eseménykezelésekhez.

#### makePolinome(inPolinome, plotFor)

Polinóm pontjainak legenerálására szolgáló függvény, a grafikon kirajzolónak megfelelő típusban

**inPolinome** polinóm



**plotFor** polinóm intervalluma és pontossága

**Connection.request(aConfig)**

Elküldi a szervernek az értékeket

**aConfig.params** a kommunikáció objektuma

**aConfig.callback** sikeres visszatérés esetén lefutó függvény

**basicTable (aConfig)**

Egy alap tábla létrehozó objektum. Ennek segítségével tudtam létrehozni az interpolációs táblázatot és a menülistát(interpoláció választó)

**that.addNewCellToRow(rowIndex, textValue, inputAttributes)**

Ad egy új cellát a sorhoz

**that.addNewRowToTable(data)**

Ad egy új sort a táblázathoz

**that.addNewColumnToTable(data)**

Ad egy új oszlopot a táblázathoz

**that.newTable()**

Új tábla létrehozása

**that.setCellForm((i , j, attributes))**

Egy adott cella megformázás beállítása

**that.getNumOfCols()**

Vissza adja az oszlopok számát

**that.getNumOfRows()**

Vissza adja a sorok számát

**that.getRow(i)**

Vissza adja a sort az index alapján. Ha nincs olyan indexű akkor null

**that.getInputTag(i, j)**

Vissza tér a tábla input elemével

**that.getValue(i, j)**

Egy adott cella érték lekérdezése

**that.findValue(column, value)**

Megkeresi melyik sorban van egy adott értéket

**that.setValue(i, j, value, form)**

Beállít egy adott értéket egy cellának

**that.deleteTable()**

Teljesen törli a táblázatot

**that.remove(row)**

Kivesz egy sort a táblázatból

**addNewRowTagToTable ()**

Ad egy új sort a táblázathoz

**addCellToRow(index)**

Ad egy cellát a sorhoz

**setAttributes(object, attributes)**

Beállítja egy objektum tulajdonságait

**makeTextInput (value, attributes)**

TextInput hozzáadása a sorhoz

**interpolationMenulist (aConfig)**

Az interpolációs menü függvénye. Itt tarjuk számon az aktuálisan betöltött adathalmazt.

**that.newItem()**

Új Lista elem

**that.getDataArray(server)**

Vissza adja az adathalmazt, tömb formában. Ebben a formában küldjük fel a szervernek.

**that.getDataObject()**

Vissza adja az adathalmazt, egy objektum formájában. Az Objektum értékeinek kulcsa, az interpolációk id-ja.

**that.saveItemSettings()**

Elmenti az adatokat az aktuálisan kijelölt sorba.

**that.loadItemSettings(index)**

Feldogozza az adott sort a táblából, és betölti az adatokat a táblába.

**that.loadAll(savedObject, resultObject)**

Betölti az összes Interpolációt az adott adathalmazból

**newMenulist()**

Új menülista: régi menü kitörlése, és egy új generálása

**interpolationPlot (aConfig)**

Grafikon megjelenítése: Flot segítségével létrehoztam az alábbi Objektumot. Ebben valósítottam meg a kirajzolást, és annak tulajdonságait.

**that.refresh(points, polinomials)**

Pontok és a polinómkok alapján frissíti a grafikont

**that.getPlotSettings**

Visszatér a grafikon megjelenítési tulajdonságokkal. Ennek segítségével mentjük el a tulajdonságokat.

**that.setPlotSettings**

Betölti a grafikon megjelenítési tulajdonságokat.

**generateData(senderData, polynomial)**

Legenrálja a grafikon azon bemenő paraméterét, amely a megjelenítendő adatokat állítja

**generateType()**

Legenrálja a grafikon azon bemenő paraméterét, amely a grafikon megjelenítését állítja

**setDefaultSettings()**

Legenrálja a grafikon azon bemenő paraméterét, amely a grafikon megjelenítését állítja

**generatePointSet(tableArray, derivNum)**

Legenerálja az adott pontokat, az interpolációs táblázatból

**interpolationTable (aConfig)**

Az interpolációs Táblázat logikája, és generálása. Ebben a táblázatban tekinthetjük meg a pontokat.

**that.addPoint(x, y, dn)**

Hozzá adja a pontot a táblázathoz. Ha létezik ezen az X-en pont akkor frissíti.

**that.setPoints(tableArray)**

Feltölti a táblázatot egy adott tömb értékeivel

**that.setData(data)**

Feltölti az adatokkal a táblát

**that.getData()**

Vissza adja a táblázatban szereplő adatok

**that.getPoints()**

Vissza adja a táblázatban szereplő pontokat

### 3.2.3. Kommunikáció

### 3.2.4. Tesztelési terv

## 3.3. Elosztott rendszer

Elosztott rendszer Erlang-ban lett megvalósítva. Az elosztást interpolációnként végezzük, vagyis annyi node-ot hozunk létre amennyi interpolációt kívánunk egyszerre kiszámítani.

### 3.3.1. Web-szerver kommunikáció

**Adat feldolgozás** Az elosztott rendszer először kap egy JSON adathalmazt melyből kinyeri a neki szükséges adatokat, és átkonvertálja.

### 3.3.2. Gép-szerver kommunikáció

### 3.3.3. Elosztás folyamata

### 3.3.4. Tesztelési terv

## 3.4. Kalkulátor

A Kalkulátor részben számítódik ki egy-egy interpolációnak az eredménye. A megkapott adatok alapján számol, ha kell létre hozza a kezdő mátrixot, kiszámolja az eredmény mátrixot, majd annak segítségével kiszámolja a polinómot.

### 3.4.1. Fontosabb számítási függvények

#### DArray interpolateMain

Kívülről meghívandó fő függvény mely elosztja és konvertálja a részeket

**DArray &x** : Az x pontok listája

**DMatrix &Y** : Az x pontokhoz tartozó y pontok halmaza

**string type** : Interpoláció típusa: lagrange, newton, hermite

**bool inverse** : Inverz interpoláció kell-e

### 3.4.2. Elosztott rendszerrel való kommunikáció

Az elosztott rendszerben hívódó számítást Erlang - erl\_nif"-el sikerült megoldanom. Az ezzel kapcsolatos dolgokat az Calculator/erlang.cpp tartalmazza.

### **3.4.3. Tesztelési terv**

A tesztelést folyamatosan végeztem a minta adatok alapján. A függvények implementálása közben ezekre a minta adatokra meghívtam, majd ezekkel számoltam. A teszteléshez a `logTest.cpp` fájlban található függvényeket alkalmaztam.

# Irodalomjegyzék

- [1] Gergó Lajos: Numerikus Módszerek, ELTE EÖTVÖS KIADÓ, 2010, [329], ISBN 978 963 312 034 7
- [2] [http://www.erlang.org/doc/man/erl\\_nif.html](http://www.erlang.org/doc/man/erl_nif.html) 2015
- [3] [https://www.sharelatex.com/learn/Sections\\_and\\_chapters](https://www.sharelatex.com/learn/Sections_and_chapters) 2015
- [4] <https://github.com/mochi/mochiweb/blob/master/src/mochijson.erl> 2015
- [5] <http://tex.stackexchange.com/questions/137055/lstlisting-syntax-highlighting-for-c-like-in-editor> 2015