



DEBRECENI EGYETEM
MŰSZAKI KAR
GÉPÉSZMÉRNÖKI TANSZÉK

UNIVERSITY OF DEBRECEN
FACULTY OF ENGINEERING
DEPARTMENT OF MECHANICAL
ENGINEERING

GUMIRUGÓ ALAKOPTIMALIZÁLÁSA
METAHEURISZTIKUS KERESŐ
ELJÁRÁSSAL
SZAKDOLGOZAT

Némethy Andor

Üzemeltető-karbantartó specializáció

Debrecen

2023

Tartalomjegyzék

Tartalomjegyzék	V
Jelölések jegyzéke	VII
1 Bevezetés	1
2 Matematikai optimalizáció.....	1
2.1 Az optimalizáció szerepe a tervezési folyamatban	2
2.2 Optimalizálási alapfogalmak	4
2.2.1 Tervezési változók, tervezési paraméterek.....	4
2.2.2 Optimalizálási kényszerek	5
2.2.3 Célfüggvény	7
2.3 Populáció alapú metaheurisztikus kereső módszerek.....	8
2.3.1 Genetikus algoritmusok.....	9
2.3.2 Hangyakolónia optimalizáció.....	9
2.3.3 Részecske-raj optimalizáció.....	10
3 Az optimalizációs feladat leírása	11
3.1 A probléma meghatározása	12
3.1.1 Tervezési változók.....	13
3.1.2 Optimalizálási kényszerek	13
3.2 Célfüggvény	14
3.2.1 Átlagos abszolút hiba.....	16
3.2.2 Átlagos négyzetes hiba négyzetgyöke	16
3.2.3 Átlagos relatív hiba.....	16
3.2.4 Az eltérések súlyozása	17
3.2.5 A választott módszer.....	17
3.3 Numerikus modell.....	17
3.3.1 DesignModeler.....	18
3.3.2 Anyagmodell	19
3.3.3 Statikus végelem szimuláció	20

3.4	Az optimalizációs probléma black box természete	21
4	Optimalizációs algoritmus	22
4.1	A részecske-raj optimalizáció részletes bemutatása	22
4.2	Sebesség függvény	24
4.3	Optimalizálási kényszerek implementálása	25
4.4	Az algoritmus pszeudokódja.....	26
4.5	Az algoritmus hiperparamétereinek megválasztása	27
4.5.1	Optimalizáció eredménye	29
4.6	Az algoritmus tesztelése	30
4.6.1	Ackley függvény	30
4.6.2	Bukin 6 függvény	31
4.6.3	Gömb függvény.....	32
4.6.4	Step függvény	33
4.6.5	Three-hump camel függvény	34
4.6.6	Tesztelés eredménye	34
5	Az optimalizáció futtatása	36
5.1	A MATLAB és Ansys közötti kapcsolat megteremtése	36
5.2	A futtatás és az eredmények kiértékelése	38
6	Összefoglalás	42
	Irodalomjegyzék	43

Jelölések jegyzéke

$\mathbf{x}$	[–]	tervezési változók vektora
x_i	[–]	tervezési változó
$\check{x}_i$	[–]	tervezési változó alsó korlátja
$\hat{x}_i$	[–]	tervezési változó felső korlátja
n_x	[–]	tervezési változók száma
$\mathbf{x}^*$	[–]	globális optimum
$f(\mathbf{x})$	[–]	célfüggvény
$g(x_i)$	[–]	egyenlőtlenség típusú optimalizálási kényszer
$h(x_i)$	[–]	egyenlőség típusú optimalizálási kényszer
n_g	[–]	egyenlőtlenség típusú optimalizálási kényszerek száma
n_h	[–]	egyenlőség típusú optimalizálási kényszerek száma
$\mathbf{x}_i$	[–]	részecske pozíciója
$\mathbf{v}_i$	$\left[\frac{1}{\text{iteráció}} \right]$	részecske sebességvektora
$\mathbf{x}_i^*$	[–]	i-edik részecske által ismert legjobb érték pozíciója
$\mathbf{g}^*$	[–]	teljes részecske-raj által ismert legjobb érték pozíciója
t	[–]	jelenlegi iteráció
T	[–]	iterációk száma
n_r	[–]	részecskék száma
c_1	[–]	kognitív együttható
c_2	[–]	szociális együttható
w	[–]	lendület
χ	[–]	<i>constriction factor</i>
D	[mm]	a gumirugó palástjának átmérője
d	[mm]	a gumirugó talpának átmérője
H	[mm]	a gumirugó magassága
α	[°]	a gumirugó oldalferdeségének szöge
$\mathbf{D}$	[N]	a keresett rugókarakterisztika pontjaiból képzett vektor

$\mathbf{S}$	[N]	a szimulált rugókarakterisztika pontjaiból képzett vektor
n_p	[–]	a vizsgálati pontok száma
ω	[–]	súly vektor

1 Bevezetés

Az optimalizáció a mindennapjaink része, gyakran anélkül is törekszünk rá, hogy észrevevénénk azt. Igyekszünk a lehető leggyorsabb útvonalon munkába jutni, vagy a boltokban a kiszemelt termékeket a lehető legolcsóbban megvásárolni. Így próbáljuk az erőforrásainkat minél kedvezőbb módon kihasználni, és életünket jobbá tenni.

Bizonyos esetekben azonban szükséges matematikai optimalizáció használata, ugyanakkor számos valós probléma rendkívül összetett, így a hagyományos módszerekkel nehéz a globális optimum megtalálása, a metaheurisztikus keresőeljárások viszont hatékonyságuk és sokoldalúságuk miatt jól alkalmazhatók megoldásukra.

A szakdolgozatom célja egy olyan robosztus optimalizálási módszer létrehozása, amely könnyen integrálható a már meglévő tervezési folyamatba, valamint minimális változtatással adaptálható bármilyen problémára. Ezt a folyamatot egy gumirugó alakoptimalizálási feladatán keresztül mutatom be, mely négy tervezési változóval és végeselemes szimulációval számítható célfüggvénnyel rendelkezik. Az eljárás felépítését két mérnöki területen népszerű szoftver segítségével valósítottam meg. Az egyik, a MATLAB, amely egy széles körben használt programnyelv, valamint numerikus számítások elvégzésére kifejlesztett környezet, a másik pedig az Ansys Workbench. Az Ansys Workbench egy átfogó rendszer, amely lehetővé teszi a felhasználók számára, hogy hozzáférjenek az Ansys által nyújtott különböző modulokhoz és funkciókhoz. Ezen optimalizálási módszer előnye az, hogy mivel mindkét szoftver népszerű az iparban, ezért a keresőeljárás jól implementálható komplex probléma esetén is.

Témaválasztáskor próbáltam arra törekedni, hogy a tradicionálisabb gépészmérnöki területeket ötvözni tudjam a programozással. Bízom benne, hogy a jövőben képes leszek ilyen jellegű pozícióban elhelyezkedni, valamint a szakdolgozatom során szerzett ismereteket kamatoztatni. Úgy gondolom, hogy az általam feldolgozott témának helye van a modern kompetitív iparban, ugyanis személyes tapasztalataim szerint nincsenek kellően kiaknázva az optimalizációban rejlő lehetőségek.

2 Matematikai optimalizáció

Az optimalizáció kifejezést gyakran használják a fejlesztés, vagy javítás szinonimájaként, holott matematikai értelemben a lehető legjobb megoldás megtalálását jelenti valamilyen szempontrendszer alapján, az irányításunk alatt lévő változók értékeinek változtatásával [1].

A matematikai optimalizáció gyökerei évezredekre nyúlnak vissza, az ókori görögökhöz. Eukleidész bebizonyította, hogy azonos kerületű négyszögek közül a négyzet rendelkezik a legnagyobb területtel, míg Hérón megfigyelte, hogy egy tükörről visszaverődő fénysugarak a legrövidebb utat választják. Az algebra megjelenése, melyet al-Khwārizmī perzsa matematikusnak tulajdonítanak, nagyban hozzájárult az optimalizálás tanulmányozásához. Maga az algoritmus kifejezés is az ő nevéből származik. A René Descartes által bevezetett koordináta-rendszer szintén kulcsfontosságúvá vált az optimalizálási problémák keresési tereinek leírásában, Gottfried Wilhelm Leibniz és Sir Isaac Newton munkássága a kalkulus kidolgozásában pedig több ma is használt optimalizációs algoritmus alapját adja [2–4].

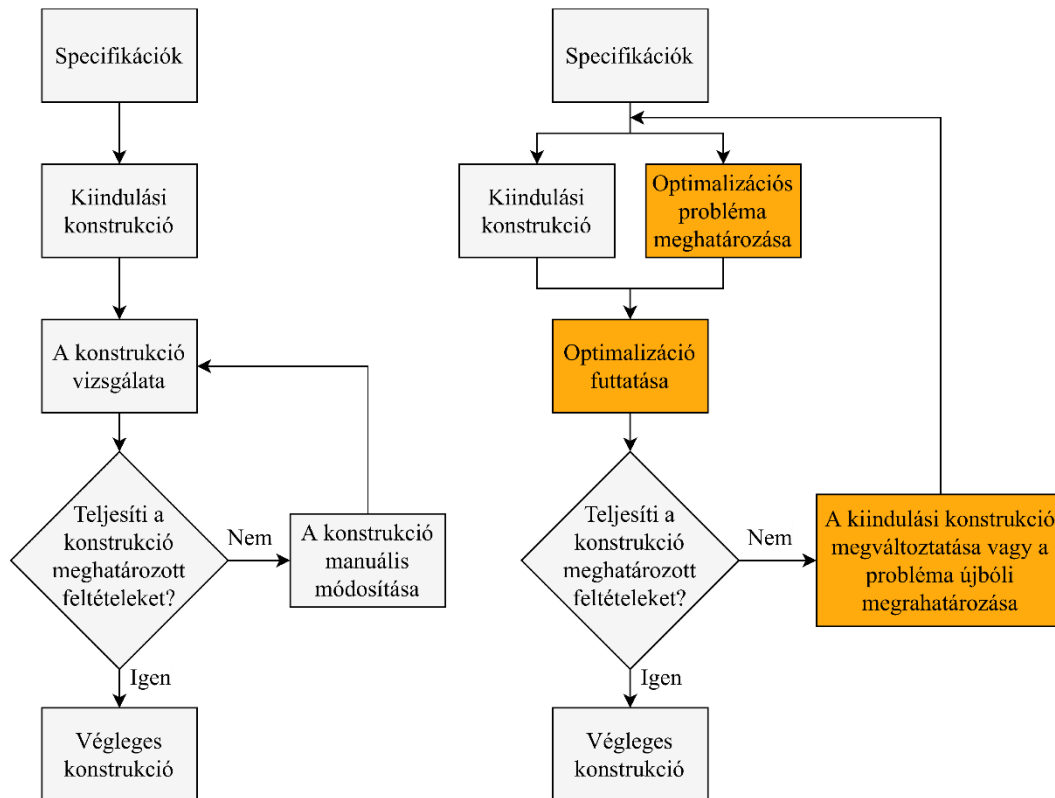
A XX. század közepén megjelentek az elektronikus számítógépek, melyek lehetővé tették, hogy az optimalizációt komplexebb problémákra is alkalmazhassák a legkülönbözőbb területeken. Az egyik nagyobb áttörést a lineáris programozás jelentette, mellyel megoldhatóvá váltak a lineáris célfüggvénnyel és lineáris optimalizálási kényszerekkel rendelkező problémák. A lineáris programozást sikeresen alkalmazták a második világháború során a rendelkezésre álló erőforrások megfelelő elosztása érdekében [2], [5].

Ugyan az egyszerűbb optimalizációs problémák analitikus módon is megoldhatók, a való életben előforduló problémák ehhez gyakran túl komplexek. A numerikus számítás és az optimalizációs algoritmusok fejlődése viszont lehetővé tette az utóbbiak megoldását is [1].

2.1 Az optimalizáció szerepe a tervezési folyamatban

Az utóbbi évtizedekben egyre elterjedtebbé vált a numerikus szimuláció, valamint 3D CAD rendszerek alkalmazása. Ezekkel az eszközökkel csökkenthető a termékek tervezési ideje, mivel így módon a költséges és időigényes fizikai kísérletezés egy része kiválthatóvá vált. Ezekhez kapcsolódó eljárás az optimalizáció, melynek segítségével tovább gyorsítható a termék elkészülésének ideje. Az optimalizáció manapság bevett eljárás, többek között a járműiparban a beépített alkatrészek nagy része valamilyen optimalizációs eljárás után nyeri el a végleges kialakítását [1], [6].

Egy valós tervezési feladat leggyakrabban nem lineáris szekvencia mentén történik, mivel egy termék tervezésekor általában számos szempontot kell figyelembe venni. Ezek kapcsolódhatnak többek között a termék gyárthatóságához, használhatóságához, gazdaságosságához és szerelhetőségéhez. A végleges konstrukciónak meg kell felelnie minden előre meghatározott követelménynek. Ahogy az 1. ábra bal oldalán is látható, sokszor előfordul, hogy meg kell ismételni a tervezési folyamat bizonyos részeit annak érdekében, hogy jobban megfeleljen a konstrukció az előzetesen meghatározott szempontoknak. Tradicionálisan a tervezési folyamat szakaszainak az ismétlése (iterálása) emberi döntések által vezérelt [1], [6].



2.1. ábra Tradicionális- és optimalizáció segítségével történő tervezési folyamat (saját szerkesztés [1] alapján)

Ahogy pedig a 2.1. ábra jobb oldala is szemlélteti, optimalizáció segítségével a tervezési folyamat jelentős része automatizálttá válik, valamint jobb konstrukciót is kaphatunk eredményül, ugyanakkor ez nem jelenti azt, hogy gombnyomásra kézhez kapjuk az optimális konstrukciót. Gyakran több szakértelmet igényel, mint a tradicionális módszer. Továbbra is szükséges emberi döntéshozatal. Meg kell határozni a specifikációkat, a kezdeti konstrukciót, az optimalizációs problémát, illetve ki kell értékelni a kapott szerkezetet. Az esetek kis százalékában lesz az első eredmény a végleges, ilyenkor a probléma újbóli meghatározása, a tervezési változók módosítása, az optimalizálási kényszerek változtatása mind segédkezhethet abban, hogy az új konstrukció jobban megfeleljen a meghatározott követelményeknek [1].

2.2 Optimalizálási alapfogalmak

2.2.1 Tervezési változók, tervezési paraméterek

A probléma leírásához szükségünk van a szerkezet véges számú mennyiség segítségével történő meghatározására. Ezen mennyiségek között megkülönböztetünk tervezési paramétereket és tervezési változókat. A tervezési paraméterek a szerkezet azon előre definiált jellemzői, melyek az optimalizáció során változatlanok maradnak, a tervezési változók pedig azok az értékek, melyeket az optimalizáció során a keresőeljárás módosítani tud [6].

A tervezési változókat az $\mathbf{x}$ oszlopvektorba rendezzük,

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_{n_x} \end{bmatrix}, \quad (2.1)$$

ahol az n_x adja meg a tervezési változók számát, vagyis a probléma dimenzióját. Az $\mathbf{x}$ vektor segítségével írjuk le az adott szerkezetet, így a különböző értékekkel rendelkező $\mathbf{x}$ vektorok, különböző egyértelműen meghatározott konstrukcióknak felelnek meg [1]. A tervezési változók által felvehető értékeket pedig a korlátok (vagy más néven határok) adják meg a következő módon:

$$\check{x}_i \leq x_i \leq \hat{x}_i \quad i = 1, \dots, n_x, \quad (2.2)$$

ahol $\check{x}$ alsó-, és $\hat{x}$ a felső korlát, melyek egy n_x dimenziós téglatestet határoznak meg [6]. A korlátok megadásakor törekedni kell arra, hogy ne csökkentsük le feleslegesen a keresési teret, így meggátolva, hogy az optimalizációs algoritmus megtalálja a lehető legjobb konstrukciót. A korlátokat valós fizikai megkötések alapján érdemes felvenni, erre példa a lemezzvastagság, egy súlycsökkentés optimalizáció során. Fontos továbbá, hogy a tervezési változók egymástól és más paraméterektől is függetlenek legyenek, ha ez nem teljesül, az optimalizációs algoritmusnak nincs abszolút irányítása a változók értékei felett [1].

A tervezési változókat általában többféleképpen is megválaszthatjuk, például egy négyzet megadható oldalhosszúságával és területével is [1]. A tervezési változókat csoportosíthatjuk, folytonos, diszkrét, illetve pszeudodiszkrét kategóriákba sorolva őket. A folytonos változók egy meghatározott intervallumon belül tetszőlegesen vehetnek fel értékeket, míg a diszkrét változók csak előre meghatározott kis számú elemekből. Utóbbit például akkor használjuk, ha azt szeretnénk eldönteni, hogy három rendelkezésre álló alapanyag közül melyikből készítsük az optimalizálandó szerkezetet. Pszeudodiszkrét változóról akkor

beszélünk, ha a tervezési változót folytonosként kezeljük, majd a feladat megoldása után kiválasztjuk a nagy számú diszkrét értékek közül a közelébe eső legjobbat [6].

2.2.2 Optimalizálási kényszerek

Az optimalizálási kényszerek (vagy más néven optimalizálási feltételek) olyan kritériumok, melyek teljesülésekor a konstrukciót „megengedettnek” tekintjük [6]. Segítségükkel a tervezési változókat korlátozzuk le valamilyen módon [1]. Egy optimalizálási kényszer lehet geometriai-, vagy működési kényszer. Geometriai kényszer esetén a tervezési változókat explicit formában, a célfüggvény meghívása előtt korlátozzuk le. Ilyen lehet például egy maximális lemezvastagság, vagy gyártástechnológiai megkötés. Működési kényszerek esetén a konstrukcióról a célfüggvény meghívása után derül ki, hogy megengedett-e vagy sem. Működési kényszert lehet felírni például a maximális feszültségre, elmozdulásra, illetve stabilitás vesztesésre. Azt a pontot, ahol minden optimalizálási kényszer teljesül a tervezési változók terén belül megengedett pontnak nevezzük, a megengedett tartomány pedig ezen pontok halmaza [6].

Az optimalizálási kényszerek leggyakrabban egyenlőtlenségek segítségével kerülnek definiálásra:

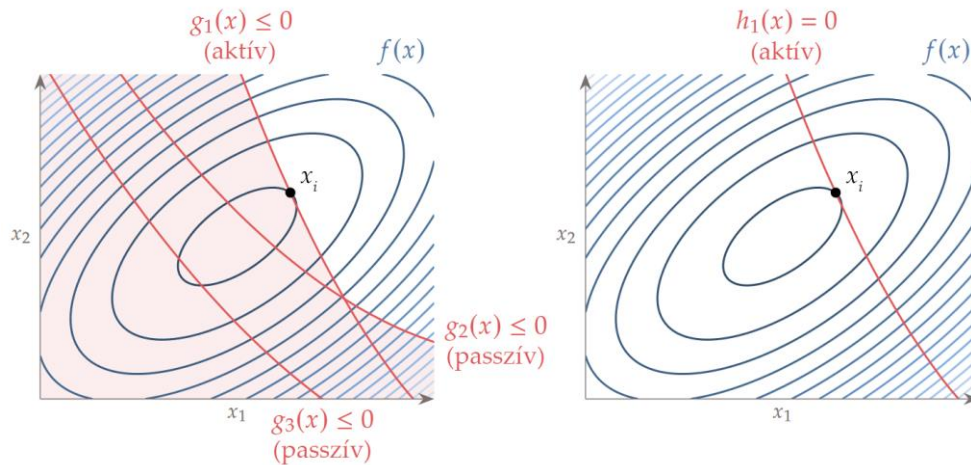
$$g_j(x_i) \leq 0 \quad j = 1, \dots, n_g; \quad i = 1, \dots, n_x, \quad (2.3)$$

ahol n_g adja meg az egyenlőtlenség típusú kényszerek számát, n_x a tervezési változók számát [1], [6]. Az egyenlőtlenség típusú kényszerek mellett szükség lehet egyenlőség típusú kényszerre is a következő alakban:

$$h_k(x_i) = 0 \quad k = 1, \dots, n_h; \quad i = 1, \dots, n_x, \quad (2.4)$$

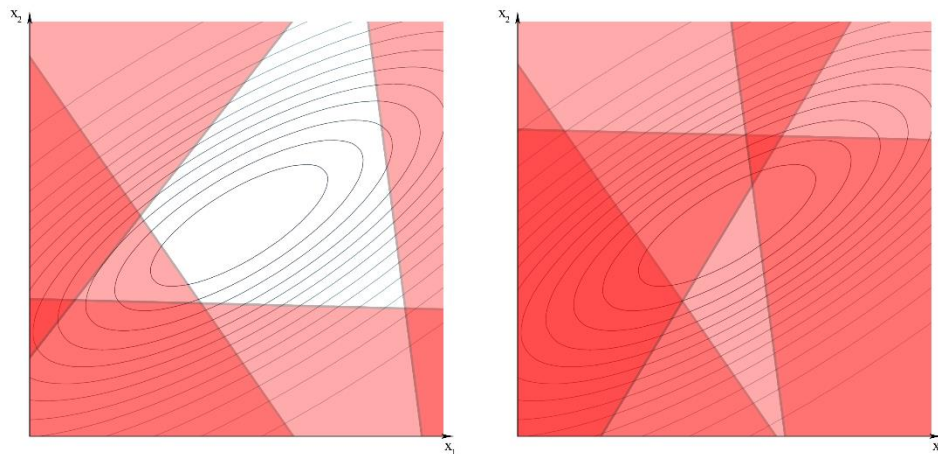
ahol n_h adja meg az egyenlőség típusú kényszerek számát. Az egyenlőség típusú kényszerek azonban kiválthatóak két egyenlőtlenség típusú kényszerrel, például $h(x) = 0$ leírható $g(x) \geq 0$ és $g(x) \leq 0$ kényszerek segítségével [1].

Az optimalizálási kényszerek lehetnek aktívak, illetve passzívak. A j -edik kényszer aktív, ha az x_i tervezési pontban teljesül a $g_j(x_i) = 0$ egyenlőség, illetve passzív, ha teljesül a $g_j(x_i) < 0$ egyenlőtlenség. Amennyiben $g_j(x_i) > 0$, az optimalizációs kényszer nem teljesül [6].



2.2. ábra Kétdimenziós optimalizációs problémák keresési terei (saját szerkesztés [1] alapján)

A 2.2. ábrán egy kétdimenziós optimalizációs probléma keresési tere látható. A bal oldalon három egyenlőtlenség típusú kényszert alkalmazunk. Ezek közül a $g_1(x)$ aktív kényszer, míg a $g_2(x)$ és $g_3(x)$ passzív. A megengedett tartomány a pirossal be nem színezett területe a keresési térnek. A jobb oldalon azt láthatjuk, hogy a $h_1(x)$ egyenlőségi kényszerrel ugyanazt az x_i eredményt tudjuk elérni [1].



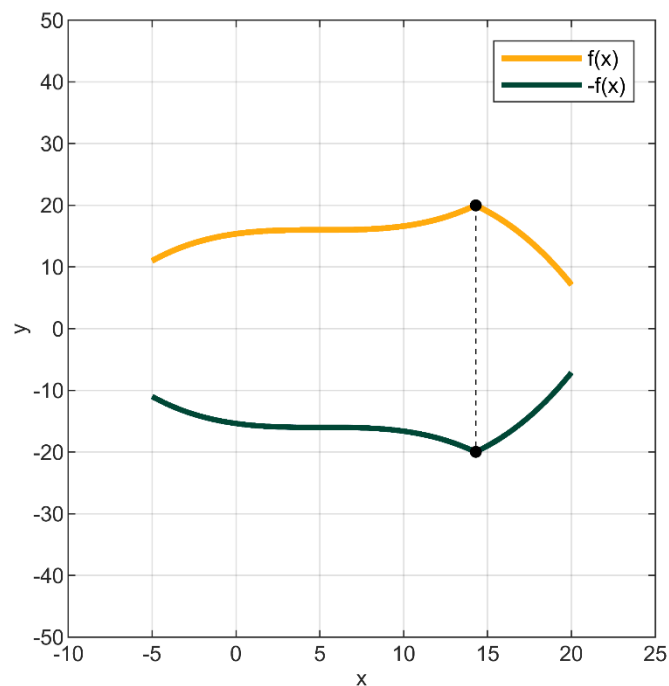
2.3. ábra Kényszerezett keresési terek megengedett tartománnyal és megengedett tartomány nélkül

A 2.3. ábrán egy kétdimenziós optimalizációs probléma keresési terét látjuk, négy pirossal jelölt kényszerrel. A bal oldalon a kényszerek úgy kerültek megválasztásra, hogy legyen megengedett tartomány. Ahogy pedig a jobb oldalon látható, egy problémát lehetséges túlkényszerezni úgy, hogy ne legyen megoldása. Ilyenkor vagy lazítani kell a kényszereken, vagy csökkenteni kell a számukat. Ezért az egyenlőség típusú kényszerek számának kevesebbnek kell lennie, mint a

tervezési változók számának. Az egyenlőtlenség típusú kényszerek esetén nincs ilyen feltétel szabva, viszont úgy kell megválasztani őket, hogy legyen megengedett tartomány [1].

2.2.3 Célfüggvény

Általában sok konstrukció eleget tesz az optimalizálási kényszereknek, viszont ahhoz, hogy ezek közül kiválaszthassuk a számunkra legjobbat, be kell vezetnünk a célfüggvényt [6]. A célfüggvény értéke skalár, mely az x pontban kiszámítható. A feladatot megfogalmazhatjuk a célfüggvény minimum – vagy maximum – pontjának keresésére való törekvésként is. Minimalizálásról beszélünk például akkor, ha egy szerkezet tömegét szeretnénk csökkenteni, míg maximalizálásról, ha egy gépjármű hatótávolságát szeretnénk növelni [1]. Az optimalizációs problémákat a leggyakrabban minimalizálásként fogalmazzuk meg, ugyanakkor ez mégsem jelent megkötést az optimalizáció univerzalitásában, mivel ahogy a 2.4. ábrán is látható, egy $f(x)$ függvény maximuma ugyanabban az x pontban van, mint $-f(x)$ minimuma [6].



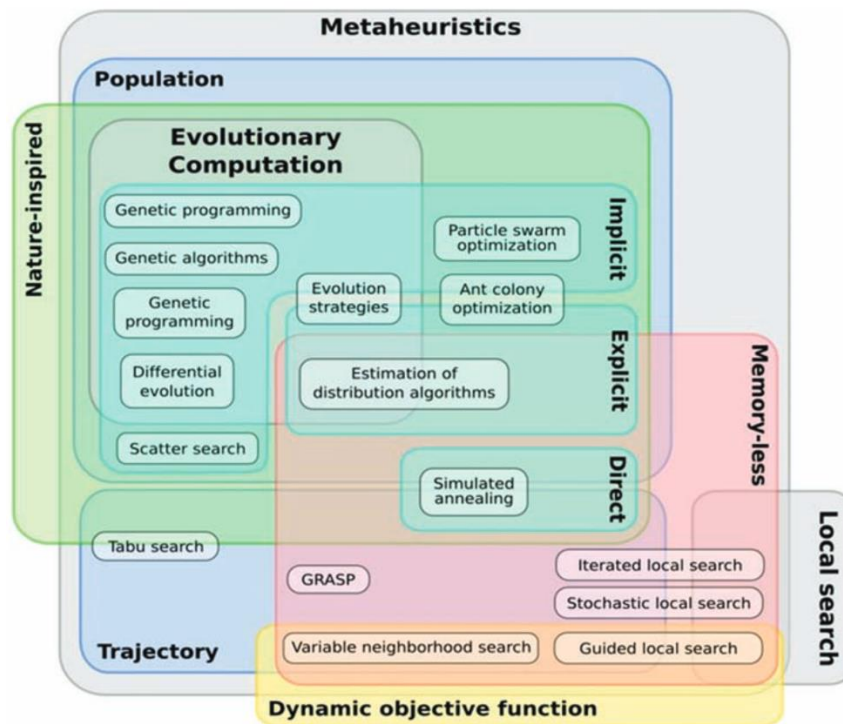
2.4. ábra Maximalizálási probléma átalakítása minimalizálásivá

A célfüggvény helyes megválasztása szükséges a sikeres optimalizációhoz. Ha a célfüggvény nem tükrözi a pontos szándékunkat, akkor a matematikai optimum nem fog mérnöki szempontból optimális konstrukciót eredményezni [1].

2.3 Populáció alapú metaheurisztikus kereső módszerek

A metaheurisztikus módszerek jellemzően magas szintű, iteratív keresőeljárások, melyek segítségével viszonylag gyorsan és hatékonyan lehet optimum-közelí eredményt meghatározni. Általános algoritmikus *framework*-öknek tekinthetők, melyek kevés módosítással sok problémára alkalmazhatók [7]. Gyakran alkalmazzák a globális- és lokális optimumkeresés valamilyen kombinációját annak érdekében, hogy effektíven fel tudja fedezni a keresési teret, mindeközben elkerülve a lokális minimumba való ragadást [8].

A metaheurisztikus módszerek kifejezetten effektívnek bizonyultak, így előszeretettel használják komplex problémák megoldására. Annak ellenére, hogy segítségükkel nem lehet garantálni a globális optimum megtalálását, mégis felül tudják múlni az egzakt módszereket, utóbbiak ugyanis gyakran nem képesek olyan megoldás meghatározására, amely minősége megközelíti a metaheurisztikus módszerekkel találhatókét [8].



2.5. ábra Metaheurisztikus keresőeljárások csoportosítása Venn-diagram segítségével [9]

Ahogy a 2.5. ábrán is látható, metaheurisztikus keresőeljárásokból rendkívül sok létezik, melyek bemutatása jócskán túlmutatna a szakirodalmi áttekintés

terjedelmén, ezért a teljesség igénye nélkül csak néhány, populáció alapú eljárást szeretnék ismertetni.

2.3.1 Genetikus algoritmusok

A genetikus algoritmusok (*genetic algorithms*, GA) Darwin-féle evolúció ihlette evolúciós algoritmusok. Az evolúció során az élőlények képesek a környezetükhöz alkalmazkodni. Ez az alkalmazkodás azonban nem ugyanolyan sikerességgel megy végbe, emiatt megjelenik szelekció is [10], [11].

A genetikus algoritmusok egyszerre több megoldással dolgoznak, ezért a megoldások populációjáról beszélhetünk. A jelenlegi populációból szelekciós operátor kiválasztja a legjobb elemeket, majd különböző egyéb operátorok segítségével új populációt állít elő. Ahhoz, hogy a jelenlegi megoldásokból az operátorok új megoldásokat tudjanak előállítani, minden megoldáshoz hozzárendelünk egy karaktersorozatot, így kódolva őket. Ezen logika mentén az optimalizáció előrehaladtával az adott populációk egyre jobb elemeket tartalmaznak [12]. Evolúciós számítások (*evolutionary computing*, EC) esetén az értékelő függvényt gyakran *fitness function*-nek (rátermettségi függvénynek) nevezzük. Az evolúciós algoritmus célja általában a *fitness function* maximalizálása, ugyanakkor az eredeti optimalizáció minimalizálási probléma is lehet, így a *fitness function* gyakran megegyezik a célfüggvénnyel, vagy transzformáltja annak [13].

2.3.2 Hangyakolónia optimalizáció

A hangyakolónia optimalizáció (*ant colony optimization*, ACO) kombinatorikai problémákra nyújt metaheurisztikus megközelítést. Az algoritmus inspirációját a hangyák által kibocsátott feromon nyomok adták, melyek kommunikációs médiumként szolgálnak. A hangyák az élelemhez vezető utat feromon csíkkal jelölik, ha több hangya járja végig ugyanazt az utat a feromon intenzitása is erősebb lesz, így jelezve az út fontosságát [11], [14].

A hangyakolónia optimalizációban a populációt egyszerű ágensek (*agents*), mesterséges hangyák alkotják, melyek mesterséges feromon nyomot használnak numerikus információ tárolására [11], [14].

Hangyakolónia optimalizációval találkozhatunk az utazó ügynök probléma (*Traveling Salesman Problem*, TSP) kapcsán. A probléma a következő: adott egy irányítás nélküli teljes gráf, melynek minden éle súlyozott, ezen pedig keressük a legkisebb összsúllyal rendelkező Hamilton kört. Az utazóügynök probléma kiemelkedően fontos a hangyakolónia optimalizáció szempontjából, mivel ez az a

probléma, amelyen először alkalmazták az első hangyakolónia alapú algoritmust, az *ant system* algoritmust, később pedig gyakran használták viszonyítási alapként új ötletek és algoritmusváltozatok tesztelésére [11], [14].

2.3.3 Részecske-raj optimalizáció

A részecske-raj optimalizáció (*particle swarm optimization*, PSO) 1995-ben lett megalkotva James Kennedy és Russel Eberhart által [15]. Az algoritmus az állatok (például madarak) rajzását veszi alapul. Az egyszerűsége és flexibilitása miatt az egyik legnépszerűbb *swarm intelligence* alapú algoritmus [16]. A *swarm intelligence* az önmagukban nem intelligens ágensek kollektíven intelligens rendszere [17].

Részecske-raj optimalizáció kontextusában egy megoldást részecskének (*particle*) nevezünk, ezek pedig rajt (*swarm*) alkotnak. Az egyes részecskék csak a saját sebességükkel, az általuk elért legjobb pozícióval $\mathbf{x}_i^*$, illetve az egész raj által elért legjobb pozícióval $\mathbf{g}^*$ vannak tisztában. Minden iteráció során a részecskék úgy változtatják a sebességüket, hogy az új pozíciójuk közelebb essen a $\mathbf{x}_i^*$ és $\mathbf{g}^*$ pontokhoz [18].

Az algoritmus jól használható neurális hálózatok tanításánál [18], elektromos hálózatok elrendezésénél, vagy akár optimalizáció segítségével történő tervezésnél. Összességében elmondható, hogy az algoritmust széles körben lehet alkalmazni. Fontos ugyanakkor megjegyezni, hogy a részecske-raj optimalizáció különösen ígéretesnek bizonyult olyan problémáknál, ahol más algoritmusok kudarcot vallottak, vagy nem áll rendelkezésre speciális keresőeljárás [19].

3 Az optimalizációs feladat leírása

A gumirugók számos alkalmazási területével találkozhatunk. Előszeretettel használják gépek, készülékek, motorok felfüggesztésére, alátámasztására, zaj- és lengéscsillapításra, valamint súlycsökkentésre. A gumirugók egyik nagy előnye a progresszív rugókarakterisztika. Nagy terhelésingadozással üzemelő járműveknél jobb választás lehet a lineáris rugókkal szemben [20].

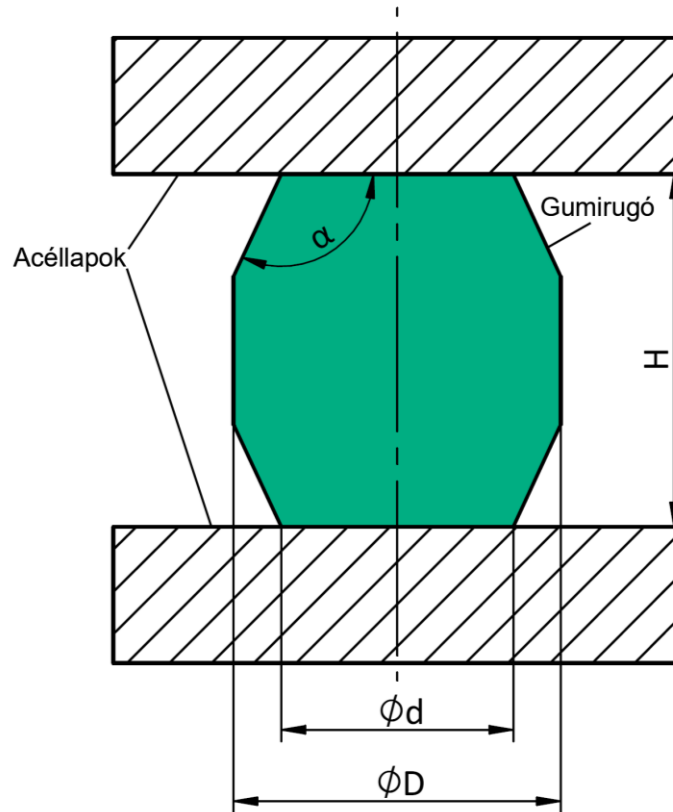
Az ipari alkalmazású gumirugók alapanyaga lehet természetes-, vagy mesterséges előállítású kaucsuk. A nagymértékű rugalmasságukat vulkanizálás útján érik el. Tulajdonságaikat a gyártási folyamat során különféle adalékok segítségével módosítják. A gumirugók anyagával kapcsolatban a legfontosabb szempontok a nagy szakítószilárdság, nagy rugalmas alakváltozás, valamint ellenállás öregedéssel és a környezeti hatásokkal szemben [20], [21].



3.1. ábra Üreges gumirugó [22]

3.1 A probléma meghatározása

Az optimalizációs probléma metszeti vázlata a 3.2. ábrán látható a tervezési változók jelölésével.



3.2. ábra. A gumirugó keresztmetszete

A gumirugó két acéllap között helyezkedik el, a lapok közelítése során a gumirugó deformálódik. A gumirugó összenyomása maximálisan 20 mm, az így ébredő rugóerő kerül kiértékelésre. A végelem szimuláció során minden második milliméteren rögzítésre kerül az adott hosszváltozáshoz tartozó rugóerő, az így kapott 10 pontot grafikonon ábrázolva pedig kirajzolódik a rugókarakterisztika. Az optimalizáció célja, hogy a végelem szimulációval kapott rugókarakterisztika **S** minél inkább megközelítse az előre meghatározott karakterisztikát **D**.

3.1.1 Tervezési változók

A gumirugó alakoptimalizálásához elengedhetetlen a változók meghatározása. Ezek a következők: a D külső átmérő, a d átmérő, a H magasság, illetve az α szög. Ezek alkotják a tervezési változók $\mathbf{x}$ oszlopvektorát:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} D \\ d \\ H \\ \alpha \end{bmatrix}. \quad (3.1)$$

A korlátok pedig a következők:

$$\tilde{\mathbf{x}} = \begin{bmatrix} 80 \text{ mm} \\ 60 \text{ mm} \\ 50 \text{ mm} \\ 93^\circ \end{bmatrix} \text{ és } \hat{\mathbf{x}} = \begin{bmatrix} 180 \text{ mm} \\ 140 \text{ mm} \\ 140 \text{ mm} \\ 160^\circ \end{bmatrix}. \quad (3.2)$$

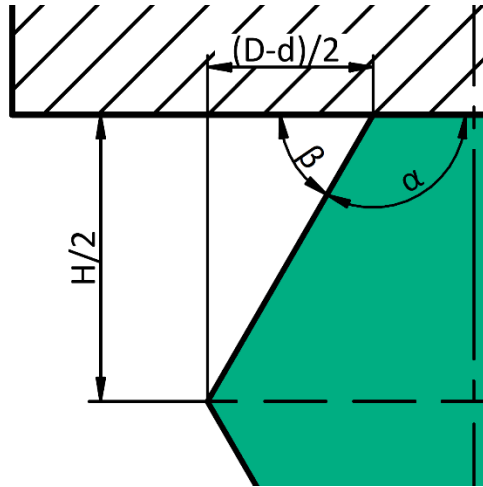
A változókat diszkrét értéktartományon választom ki, mivel mérnöki szempontból nem érzem szükségesnek 1 mm-nél és 1°-nál kisebb mértékegységek használatát, tekintve, hogy a mintának választott termék gyártói oldalán [22] jelölt DIN 3302 M4 tűrés ennél jóval durvább [23]. Ezzel a megközelítéssel továbbá az optimalizáció futásidejét is le lehet csökkenteni. Folyamatos változókat használva az optimalizáció végső szakaszában, lokális kereséskor több részecske is egymáshoz közeli értékeket vesz fel. Amennyiben pedig véges számú egész szám közül választjuk meg a tervezési változók értékeit, akkor egy adott pont egyszer már kiszámolt értékét el tudjuk tárolni, és későbbi ciklusokban a végeelem szimuláció meghívása nélkül tudunk vele dolgozni. A végeelem szimuláció numerikus jellegét tekintve szintén nem látom értelmét folyamatos tervezési változók alkalmazásának.

3.1.2 Optimalizálási kényszerek

Mivel a tervezési változók értékeit meg lehet úgy választani a korlátokon belül, hogy ne lehessen megszerkeszteni a metszeti geometriát, ezért a D külső átmérő és H magasság kiválasztását követően optimalizálási kényszereket kell felírni a d átmérőre, valamint az α szögre. A d átmérő értékét válasszuk meg úgy, hogy legyen jelentősebb méretkülönbség a kisebb és nagyobb átmérő között:

$$d \leq D - 6. \quad (3.3)$$

Ahogy a 3.3. ábrán is látható, a lehető legnagyobb oldalferdeség esetén (ha a ferdeség által kirajzolt derékszögű háromszög egyik befogójának mérete $H/2$) a β határszög a következő:



3.3. ábra A gumirugó megengedett legnagyobb oldalferdesége

$$\beta = 180^\circ - \alpha \quad (3.4)$$

$$\operatorname{tg} \beta = \frac{\frac{1}{2}H}{\frac{1}{2}(D-d)} = \frac{H}{D-d} \quad (3.5)$$

$$\beta = \operatorname{arctg}\left(\frac{H}{D-d}\right) \quad (3.6)$$

$$\alpha = 180^\circ - \operatorname{arctg}\left(\frac{H}{D-d}\right) \quad (3.7)$$

Mivel a fenti egyenletben meghatározott α szög a lehető legkisebb szög, ami mellett még kirajzolható az oldalferdeség, ezért az optimalizálási kényszer a következő:

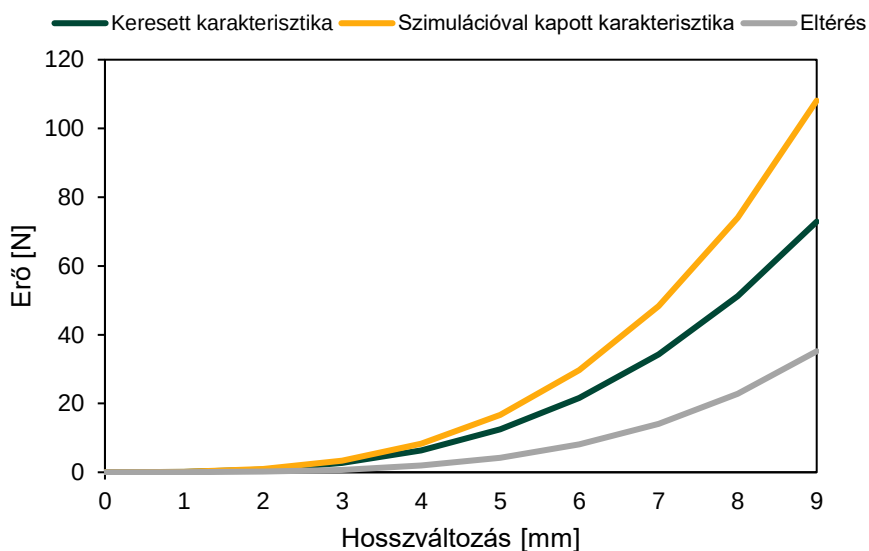
$$\alpha > \left[180^\circ - \operatorname{arctg}\left(\frac{H}{D-d}\right) \right] \quad (3.8)$$

A felfele kerekítésre azért van szükség, hogy egy esetleges számítási- vagy kerekítési pontatlanság, ne eredményezzen hibát a geometria generálásakor.

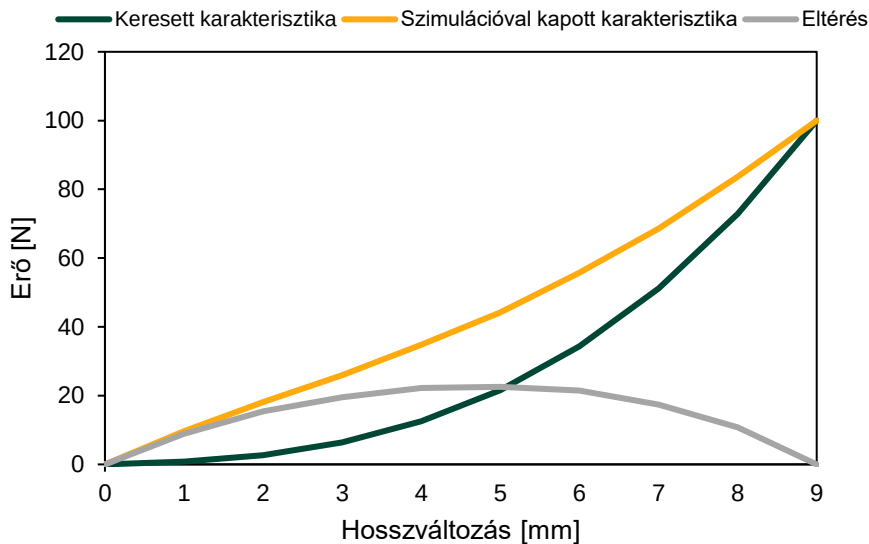
3.2 Célfüggvény

Az optimalizáció célja, hogy a szimulációval kapott gumirugó karakterisztikája minél inkább megegyezzen, az előzetesen kijelölt rugókarakterisztikával. Ez a gyakorlatban a két rugókarakterisztika közötti eltérés minimalizálását jelenti. A rugókarakterisztika hosszváltozás-erő függvényen kerül ábrázolásra, a grafikonon felvett pontok pedig kirajzolják a gumirugó jelleggörbét.

A két görbe hasonlósága azonban többféleképpen is értékelhető. A 3.4., illetve a 3.5. ábrán a két görbe közötti eltérés az átlagos abszolút hiba módszerrel került meghatározásra a (3.11) képletnek megfelelően.



3.4. ábra Progresszív-progresszív jelleggörbék összehasonlítása



3.5. ábra Progresszív-kvázilineáris jelleggörbék összehasonlítása

Az átlagos abszolút hiba módszerrel kapott célfüggvények értékei mindkét esetben egyenlők, azonban az egyes esetekben vizsgált rugók jelleggörbéi jelentősen különböznek egymástól. Az értékelés során fel kell tennünk a kérdést, hogy milyen tulajdonságok fontosabbak a számunkra, ugyanis ez alapján tudunk meghatározni egy olyan célfüggvényt, ami ezeket előnyben részesíti.

A két rugókarakterisztika közötti eltérés számítására több eljárás is használható, a módszert azonban valóéletbeli szempontok alapján kell megválasztani. Fontosabb lehet például az, hogy minél inkább hasonlítson a két rugó jelleggörbéje (legyen az lineáris, progresszív vagy degresszív), vagy akár az, hogy egy bizonyos hosszváltozáshoz adott erő legyen szükséges.

Az eltérés számítására jól alkalmazhatók a statisztikai analízisben használt, előrejelzés pontosságát megadó összefüggések. Ebben a kontextusban előrejelzési hibáról (*forecast error*) beszélünk. Az előrejelzési hiba megadására számos módszer létezik, ilyen például a korábban említett átlagos abszolút hiba is [24].

3.2.1 Átlagos abszolút hiba

Talán a legkézenfekvőbb a korábban már említett átlagos abszolút hiba (*Mean Absolute Error*, MAE) módszer alkalmazása:

$$f(x) = \frac{1}{n_p} \cdot \sum_{i=1}^{n_p} |D_i - S_i|, \quad (3.9)$$

ahol D_i a keresett rugóerő az i -edik vizsgálati pontban, S_i a szimuláció során kinyert rugóerő az i -edik vizsgálati pontban, n_p pedig a vizsgálati pontok száma [25], [26]. Ezzel a módszerrel a célfüggvény minimuma 0.

3.2.2 Átlagos négyzetes hiba négyzetgyöke

Átlagos négyzetes hiba négyzetgyöke (*Root Mean Square Error*, RMSE) módszer alkalmazása:

$$f(x) = \sqrt{\frac{1}{n_p} \cdot \sum_{i=1}^{n_p} (D_i - S_i)^2}, \quad (3.10)$$

ahol D_i a keresett rugóerő az i -edik vizsgálati pontban, S_i a szimuláció során kinyert rugóerő az i -edik vizsgálati pontban, n_p pedig a vizsgálati pontok száma [25], [26].

3.2.3 Átlagos relatív hiba

Átlagos relatív hiba (*Mean Absolute Percentage Error*, MAPE) módszer. A továbbiakban a dolgozatban átlagos relatív hiba alatt a következő összefüggést értem:

$$f(x) = \frac{1}{n_p} \cdot \sum_{i=1}^{n_p} \frac{|D_i - S_i|}{D_i} \quad (3.11)$$

ahol D_i a keresett rugóerő az i -edik vizsgálati pontban, S_i a szimuláció során kinyert rugóerő az i -edik vizsgálati pontban, n_p pedig a vizsgálati pontok száma [27]. Ez a módszer relatív hibaértéket eredményez.

3.2.4 Az eltérések súlyozása

Előfordulhat, hogy egy vagy több vizsgálati pontban fontosabb, hogy a két karakterisztika közel essen egymáshoz. Ebben az esetben használhatunk egyedi súlyozást is. Ebben az esetben bevezetünk egy n_p elemszámú ω vektort.

$$\omega = (\omega_1, \omega_2, \dots, \omega_n) \quad (3.12)$$

A súlyozást bármelyik felsorolt vizsgálati módszerrel lehet használni. Átlagos abszolút hiba módszerrel a következő:

$$f(x) = \frac{1}{n_p} \cdot \sum_{i=1}^{n_p} (\omega_i \cdot |D_i - S_i|) \quad (3.13)$$

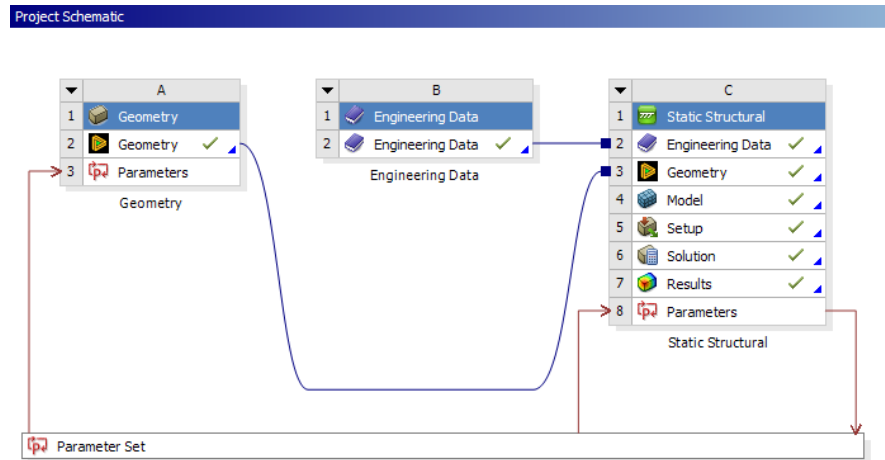
3.2.5 A választott módszer

Módszerválasztáskor fontos figyelembe venni a problémát. Felvetődhet például a kérdés, hogy adott vizsgálati pont 20 N-os eltérése kétszer olyan rossz-e, mint a 10 N-os eltérése; a pozitív és negatív eltérés ugyanolyan jelentőségű-e; vagy, hogy a karakterisztika jellege mennyire fontos számunkra. Mindezen kérdések valós szempontokon alapulnak, mely szempontok pedig más-más eltérés értékelési módszereket követelnek meg. Számos módszert kreálhatunk a különböző tervezési szempontok alapján, viszont valós ipari probléma hiányában ilyen szempontok jelen esetben nem állnak rendelkezésre.

Személy szerint az átlagos relatív hiba módszert tartom a legkézenfekvőbb választásnak a célfüggvény implementálásához, ezért ezt fogom a továbbiakban alkalmazni. Ugyanakkor további kutatás témáját képezheti a különböző módszerek összehasonlítása és értékelése.

3.3 Numerikus modell

A numerikus modellt ANSYS Workbench segítségével építettem fel. A Workbench környezetben integráltan megtalálható és használható az összes ANSYS modul. A 3.6 ábrán látható a szimuláció felépítése.



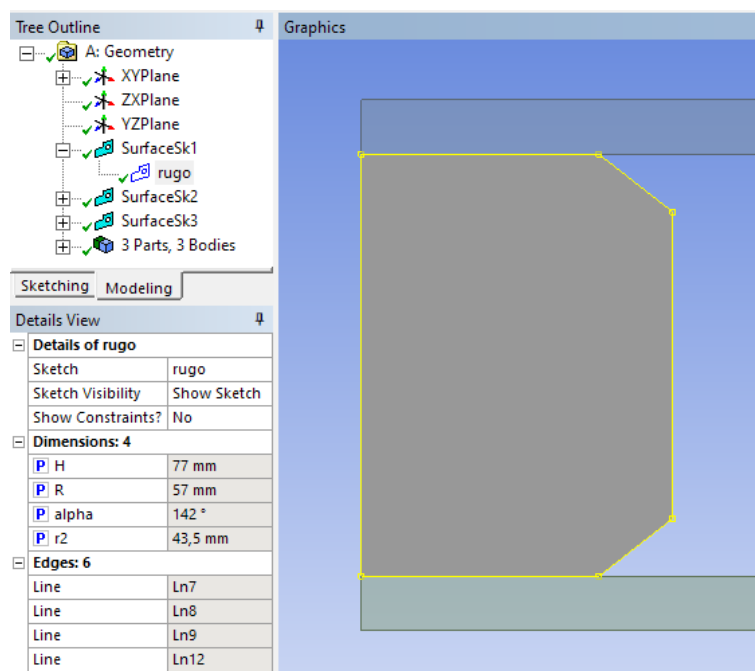
3.6. ábra A szimuláció felépítése az ANSYS Workbench környezetben

A Geometry modulban kerül felépítésre és parametrizálásra a rugógeometria, az Engineering Data tartalmazza az anyagi jellemzőket, a Static Structural modul végzi el a szimulációt, a Parameter Set pedig tárolja a bemeneti és kimeneti változókat.

3.3.1 DesignModeler

A rugó CAD modellje a DesignModeler segítségével került felépítésre, így az optimalizációs algoritmus képes módosítani a tervezési változók segítségével a rugó geometriáját. Ebből kifolyólag a tervezési változókat parametrizált értékként kell létrehozni. Fontos megjegyezni, hogy a SpaceClaim segítségével van mód tervezőprogramban kreált CAD modell importálására és parametrizálására is, ugyanakkor jelen esetben kézenfekvőbb a geometria megrajzolása a DesignModeler környezetben. A parametrizálás az egyetlen módosítás, amit egy valós probléma esetén ejteni kell a felépített mechanikai modellen annak érdekében, hogy az optimalizáció alkalmazható legyen.

Forgásszimmetrikus egyszerűsítéssel felépített szimuláció esetén a forgástengely az ANSYS megoldó esetén az Y tengely, a keresztmetszet felét így az X^+ és Y tengely által határolt félsíokban kell létrehozni, így viszont a parametrizált értékek a tervezési változóktól eltérnek: D és d átmérők helyett R és r rádiuszokkal van meghatározva a geometria. Ezt észben tartva az optimalizáció során az átmérők értékeit át kell váltani a végelem szimuláció meghívása előtt.



3.7. ábra A gumirugó parametrizált keresztmetszete DesignModeler környezetben

Az így létrehozott 4 paraméter érték már módosítható a DesignModeler modulon kívül, a Workbench környezetben is. Az értékük átírása után pedig újra lehet generálni a rugó geometriáját, ami aztán automatikusan importálható a további modulokba. A 3.8. ábra illusztrálja a parametrizált bemeneti értékeket.

2	Input Parameters			
3	Geometry (A1)			
4	P1	rugoh	77	mm
5	P2	rugor	57	mm
6	P3	rugalpha	142	degree
7	P4	rugokisr	43,5	mm

3.8. ábra Bemeneti paraméterek

3.3.2 Anyagmodell

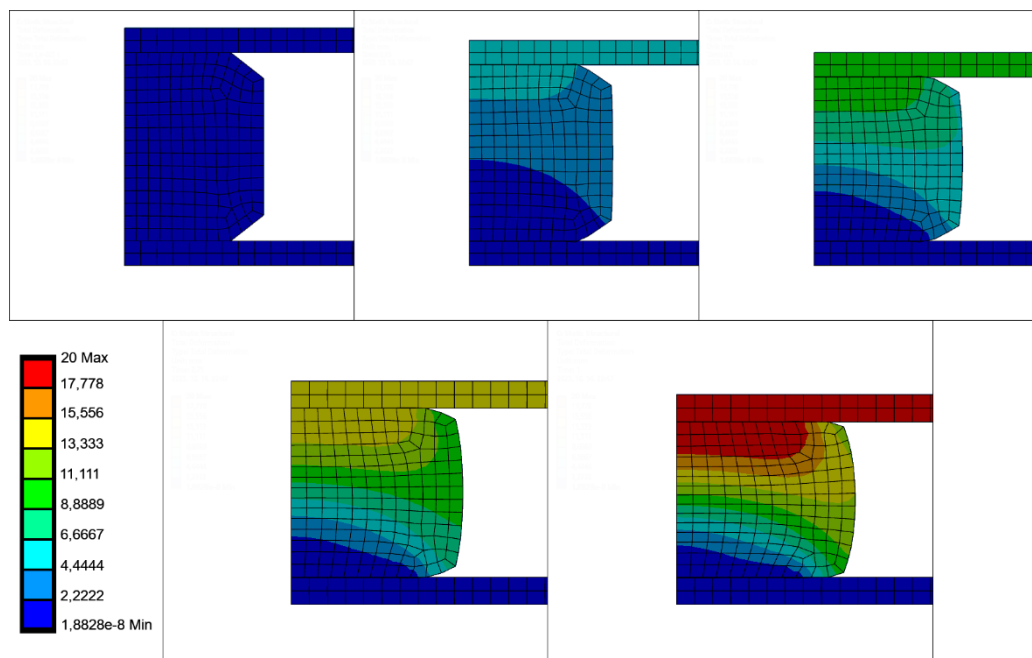
Az anyagmodell létrehozásakor hasonló témában végzett kutatást [28] vettem alapul. Tekintettel arra, hogy nem gyakorlati problémáról van szó, optimalizációs eljárás gyakorlati demonstrációja esetén elegendő ezen modell használata.

	A	B	C	D	E
1	Property	Value	Unit		
2	Yeoh 3rd Order			☐	
3	Material Constant C10	1,6117	MPa		☐
4	Material Constant C20	0,46206	MPa		☐
5	Material Constant C30	0,45899	MPa		☐
6	Incompressibility Parameter D1	0,002	MPa ⁻¹		☐
7	Incompressibility Parameter D2	0,002	MPa ⁻¹		☐
8	Incompressibility Parameter D3	0,002	MPa ⁻¹		☐

3.9. ábra Az alkalmazott anyagmodell és paraméterek

3.3.3 Statikus végelem szimuláció

Hozzá lett rendelve a rugóhoz a korábban felvett gumi anyagmodell, valamint a két laphoz az Ansys alapértelmezett *Structural Steel* anyagmodellje. A gumirugó és a két lap között egy-egy *frictional contact* került felvételre 0,6-os súrlódási tényező mellett. Ez az érték helytálló acél és gumi közötti érintkezéskor [29]. A hálózás 5 mm-es elemmérettel történt, ez jóval kisebb, mint a lehető legkisebb konstrukció méretei, így a tervezési változók bármilyen értékei mellett elvégezhető marad. Az analízis 100 lépésben került megoldásra, a kiértékelés pedig minden tizedik ponton történt. A 3.10. ábrán a gumirugó különböző mértékű összenyomódásai láthatók, a színskála pedig a deformációt jeleníti meg.



3.10. ábra A gumirugó elmozdulás mezője különböző összenyomódási mértékek esetén

3.4 Az optimalizációs probléma *black box* természete

Az optimalizációs probléma a végelem szimulációból kifolyólag felfogható egy *black box* függvényként: Ahogy a 3.11. ábrán is látható beszélhetünk bemenetekről és kimenetekről. Bemenet jelen esetben a tervezési változók vektora, kimenet pedig a célfüggvény visszatérési értéke.



3.11. ábra Az optimalizáció *black box* természete

A *black box* a bemenetektől valamilyen folyamat során kimenetet csinál. Az optimalizációs példa esetében ez a folyamat magába foglalja a végelem szimulációt, valamint a mérési pontok kiértékelését. A célfüggvény így nem írható fel analitikus módon, az optimalizáció *black box* természete a numerikus szimuláció alapegyenletéből, valamint a hibafüggvényből ered.

4 Optimalizációs algoritmus

Keresőeljárás választásakor fontos figyelembe venni a probléma jellegét. A végeselem szimuláció alapegyenletének numerikus megoldása miatt úgynevezett *black box* függvény optimalizációjáról beszélhetünk. Ilyen probléma esetén anélkül kell megtalálnunk a függvény minimumpontját, hogy ismernénk a függvény viselkedését, valamint a függvény lehetséges értékeit. Ebben az esetben a keresőeljárásnak két tulajdonsággal kell rendelkeznie. Egyrészt képesnek kell lennie a keresés közbeni tanulásra: információt kell gyűjtenie a keresési térről, és az ígéretes régiókra kell koncentrálnia a keresést. Másrészt az algoritmusnak képesnek kell lennie a tartós keresésre, vagyis, ha a legígéretesebb régiókban végzett keresés nem talál kielégítő pontot, az algoritmusnak a keresési tér más régióiban kell folytatnia a keresést. Így az utóbbi évtizedekben kifejlesztett metaheurisztikus eljárások közül a részecske-raj optimalizációra esett a választásom népszerűsége és sokoldalúsága miatt. Sokat taglalt, valamint számos szakirodalom foglalkozik a módszerrel, így nem okozott gondot az algoritmus kutatása [30].

4.1 A részecske-raj optimalizáció részletes bemutatása

A részecske-raj optimalizáció a keresési teret mozgó pontok, úgynevezett részecskék segítségével fedezi fel, kvázi-sztochasztikus módon. A részecskék összességét rajnak nevezzük. Az algoritmus T számú ciklusból áll, melyek során a részecskék pozíciója a sebességfüggvény segítségével kerül kiszámításra. A sebességfüggvény két fő komponensből áll: egy determinisztikus-, és egy sztochasztikus komponensből. A determinisztikus komponens magában foglalja a részecskék globális (egész raj által ismert) legjobb $\mathbf{g}^*$ pozícióhoz, valamint az adott részecske által ismert legjobb $\mathbf{x}_i^*$ pozícióhoz való tartását. Mindeközben a sztochasztikus komponens hatására mozgásuk bizonyos mértékig véletlenszerűvé válik [16].

A részecske sebességét tradicionálisan a következő módon kapjuk meg:

$$\begin{aligned} \mathbf{v}_i(t+1) = w\mathbf{v}_i(t) + c_1\mathbf{r}_1(t)[\mathbf{x}_i^*(t) - \mathbf{x}_i(t)] + \\ + c_2\mathbf{r}_2(t)[\mathbf{g}^*(t) - \mathbf{x}_i(t)], \end{aligned} \quad (4.1)$$

ahol $\mathbf{v}_i(t)$ az i -edik részecske sebessége a t időpillanatban, w a lendület, ami megadja, hogy az i -edik részecske $\mathbf{v}_i(t-1)$ sebessége mekkora hatással van a $\mathbf{v}_i(t)$ sebességre, $\mathbf{x}_i(t)$ az i -edik részecske pozíciója a t időpillanatban, c_1 és c_2 a kognitív- és szociális együttthatók ebben a sorrendben, $\mathbf{r}_1(t)$ és $\mathbf{r}_2(t)$ pedig egyenletes eloszlású ciklusonként véletlenszerűen választott számok a $[0; 1]$ intervallumon. Az $\mathbf{r}_1(t)$ és $\mathbf{r}_2(t)$ alkotja az algoritmus sztochasztikus komponensét [31], [32].

Az $\mathbf{x}_i^*$ jelöli az i -edik részecske által ismert legjobb pozíciót, míg $\mathbf{g}^*$ a teljes raj által ismert legjobb pozíciót. Ezen változók értékei az összes ciklus alatt változhatnak. Globális minimum keresésekor az i -edik részecske által ismert legjobb pozíció értéke változik, ha

$$\mathbf{x}_i^*(t-1) > \mathbf{x}_i(t) \quad (4.2)$$

és így

$$\mathbf{x}_i^*(t) = \mathbf{x}_i(t). \quad (4.3)$$

Továbbá változik a teljes raj által ismert legjobb pozíció értéke is, ha

$$\mathbf{g}^*(t-1) > \mathbf{x}_i(t) \quad (4.4)$$

és így

$$\mathbf{g}^*(t) = \mathbf{x}_i(t). \quad (4.5)$$

A részecske-raj optimalizáció során fontos egyensúlyt találni az *exploration* (felfedezés) és *exploitation* (kiaknázás) között. Az *exploration* jellemzően az optimalizáció kezdeti szakaszában preferált, ugyanakkor a megfelelő eredmény érdekében elengedhetetlen, hogy érvényesülni tudjon az *exploitation* is. A raj előbbi viselkedését nevezhetjük globális keresésnek is, mely segítségével az algoritmus felfedezi a keresési teret, míg az utóbbit lokális keresésnek, ami során a már megtalált minimumpont környékét fedezi fel [32], [33].

A c_1 és c_2 együttthatók súlyozzák a sebességfüggvény kognitív-, és szociális komponensét. Ezen két együtttható egymáshoz viszonyított aránya jelentősen befolyásolja a részecskék viselkedését. Ha $c_1 = c_2 = 0$ és $w = 1$, akkor a részecskék a jelenlegi sebességükkel, irányváltoztatás nélkül mozognak a keresési térben, ha pedig $c_1 > 0$ és $c_2 = 0$, a részecskék egymástól függetlenül mozognak. Ebben az esetben minden egyes részecske lokális keresést végez. Ugyanakkor, ha

$c_2 > 0$ és $c_1 = 0$, akkor az egész rajt a globálisan ismert legjobb $\mathbf{g}^*$ pozíció vonzza [31].

A részecskék effektivitása az együttműködésből ered. A tapasztalat azt mutatja, hogy akkor a leghatásosabbak, ha a kognitív- és szociális együtttható között egyensúlyt találunk, például, ha $c_1 \approx c_2$. Gyakran a két együtttható egymással egyenlő, viszont érdemes az értéküket probléma specifikusan megválasztani. Ha $c_1 \gg c_2$, az egyes részecskék sokkal inkább vonzódnak a saját maguk által ismert legjobb pozícióhoz, több felfedezést eredményezve. Ugyanakkor, ha $c_2 \gg c_1$, a részecskéket jobban vonzza a globálisan ismert legjobb pozíció, így idő előtt ide csoportosulnak. Egy minimumponttal rendelkező problémánál a nagyobb szociális konstans előnyösebb, míg több lokális minimumponttal rendelkező probléma esetén a nagyobb kognitív konstans [31].

4.2 Sebesség függvény

Az algoritmus tesztelése során azt tapasztaltam, hogy a részecskék nem konvergálnak a globális minimumpont felé. A sebesség skaláris nagysága gyakran meghaladta a keresési teret, vagyis egy iteráció alatt ezen kívülre kerültek a részecskék. A megoldást egy úgynevezett *constriction factor* bevezetése jelentette [34], [35]:

$$\chi = \frac{2\kappa}{|2 - \phi - \sqrt{\phi^2 - 4\phi}|}, \quad (4.6)$$

ahol χ a *constriction factor*, $\phi = c_1 + c_2$ és $\kappa \in [0; 1]$ [35]. Az egyenletben a κ paraméter befolyásolja az *exploration* és *exploitation* közötti egyensúlyt. Ha $\kappa \approx 0$, akkor a gyors konvergencia a lokális kereséssel kerül elérésre, ugyanakkor, amennyiben $\kappa \approx 1$, akkor a konvergencia lassabb, jelentősebb a keresési tér felfedezése. A *constriction factor* előnye, hogy ha $\phi \geq 4$ és $\kappa \in [0; 1]$, akkor a rajt garantáltan konvergálni fog [31]. Hatására a részecskék az optimalizáció előrehaladtával, egyre inkább közelednek a minimum pont felé. Így a ténylegesen alkalmazott sebességfüggvény a következő:

$$\begin{aligned} \mathbf{v}_i(t+1) = & \chi \{ \mathbf{v}_i(t) + c_1 \mathbf{r}_1(t) [\mathbf{x}_i^*(t) - \mathbf{x}_i(t)] + \\ & + c_2 \mathbf{r}_2(t) [\mathbf{g}^*(t) - \mathbf{x}_i(t)] \}. \end{aligned} \quad (4.7)$$

4.3 Optimalizálási kényszerek implementálása

A korábban felírt optimalizálási kényszerek garantálják, hogy a tervezési változók bármilyen választott értékei mellett megszerkeszthető a gumirugó geometriája. Fontos továbbá, hogy a kényszerek ellenőrzése a célfüggvény meghívása előtt legyen elvégezve, ugyanis a szimuláció csak így képes lefutni.

A tényleges megvalósítás a következő: minden ciklusban a részecskék új pozíciójának a megválasztása után a tervezési változók ellenőrzésen esnek át. Ha például egy változó értéke meghaladja a felső korlátot, akkor az korrigálásra kerül a felső korlát értékére. Ugyanezen logika mentén járunk el optimalizálási kényszerek esetében is. Így a célfüggvény következő meghívásakor már a tervezési változók korrigált értékeivel történik a számítás.

4.4 Az algoritmus pszeudokódja

Az algoritmus működése ún. pszeudokód segítségével is szemléltethető. A pszeudokód egy egyszerűsített, jobban átlátható nyelv a programok leírására. Lehetővé teszi az olvasó számára, hogy az algoritmus logikájára koncentráljon a programnyelv szintaktikája helyett [36]. Így az algoritmus a következő:

```
ciklusszám definiálása
részeckszám definiálása
korlátok definiálása

for i = 1:nx //minden részecskére nézve
    kezdeti részecske pozíciók véletlenszerűen választva
        egyenletes eloszlás szerint:  $\hat{x} \leq x_i(0) \leq \hat{x}$ 
     $v_i(0) = 0$ 
end

for t = 1:T //minden ciklusban
    for i = 1:nx
        if  $f[x_i(t)] < f(x_i^*)$  //  $x_i^*$  és  $g^*$  értékének frissítése
            részecske által ismert legjobb pozíció
            frissítése:  $x_i^* = x_i(t)$ 
        if  $f[x_i(t)] < f(g^*)$ 
            raj által ismert legjobb pozíció
            frissítése:  $g^* = x_i(t)$ 
        end
    end
end
for i = 1:nx
    sebesség kiszámítása:  $v_i(t+1) =$ 
         $= \chi\{v_i(t) + c_1 r_1(t)[x_i^*(t) - x_i(t)] + c_2 r_2(t)[g^*(t) - x_i(t)]\}$ 
    új pozíciók kiszámítása egész számra
        kerekítéssel:  $x_i(t+1) = \lfloor x_i(t) + v_i(t+1) \rfloor$ 
end
for i = 1:nx
    korlátok, optimalizálási kényszerek
        betarttatása (pozíciókorrigálás)
end
end
```

4.1. forráskód A részecske-raj optimalizáció pszeudokódja

4.5 Az algoritmus hiperparamétereinek megválasztása

A hiperparaméterek (c_1 , c_2 , n_r , T , κ) értékeinek közvetlen hatása van az optimalizáció sikerességére. Ugyanakkor a paraméterek változtatása nem szemléletesen befolyásolja az algoritmus futását, így kézenfekvő, hogy ezeket az értékeket optimalizáció segítségével határozzuk meg, így növelve az esélyt kielégítő eredmény találására.

A c_1 , c_2 együtthatók a részecskék viselkedését befolyásolják, míg az n_r részecske- és a T ciklusszám pedig hatással van a futásidőre. Egyszerű matematikai célfüggvény esetén gyors (akár 10 μ s-os nagyságrendű is lehet) a függvényérték meghatározása, így megengedhető a függvény sokszori meghívása, vagyis a nagy részecske- és ciklusszám. Végeselem szimuláció esetén viszont a célfüggvény egyszeri meghívása is perceket vesz igénybe, ezért kritikus ennek a minimalizálása. A gumirugó optimalizációjához előzetesen 5 órás futásidőt jelöltem ki. Ezen időtartam áll rendelkezésre a végleges konstrukció meghatározására, így korlátozva azt is, hogy a végeselem szimulációt hányszor lehet lefuttatni. Mivel a szimuláció egyszeri futtatása nagyjából 90 másodperc, ezért 200 alkalommal lehet elvégezni. A futtatások-, a részecskék- és a ciklusok száma között a következő összefüggés teremt kapcsolatot:

$$\text{szimuláció összes futtatása} = n_r \cdot T. \quad (4.8)$$

A 4.8 egyenletből kifolyólag elég csak a részecskék számát felvenni tervezési változóként, mivel ez egyértelműen megadja a maximális ciklusok számát is. Így a tervezési változók vektora a következő:

$$\mathbf{x} = \begin{bmatrix} c_1 \\ c_2 \\ n_r \\ \kappa \end{bmatrix}, \quad (4.9)$$

ahol

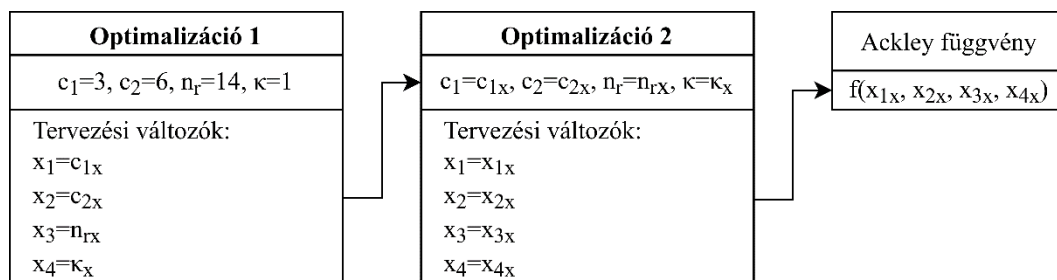
$$\tilde{\mathbf{x}} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} \text{ és } \hat{\mathbf{x}} = \begin{bmatrix} 10 \\ 10 \\ 100 \\ 1 \end{bmatrix}, \quad (4.10)$$

valamint

$$c_1 \in \mathbb{N}, \quad c_2 \in \mathbb{N}, \quad n_r \in \mathbb{N} \text{ és } K \in \mathbb{R}. \quad (4.11)$$

Optimalizáció során fontos, hogy minél kisebb értékeket találjon az algoritmus, ugyanakkor szintén fontos, hogy konzisztensen tegye ezt. Képesnek kell lennie

reprodukálható módon működni, mivel ez biztosítja azt, hogy nem véletlenül talált minimumról van szó, hanem valamilyen megismételhető módszer eredménye. Így kaphatunk olyan megoldást, ami megbízható, vagyis nem kell attól tartani, hogy az optimalizáció újbóli futtatása jelentősen jobb eredményhez vezetne.



4.1. ábra A hiperparaméterek optimalizációja

A hiperparaméterek megválasztásának folyamatát a 4.1. ábra szemlélteti, a módszer pedig a következő: A tetszőlegesen megválasztott hiperparaméter értékekkel rendelkező 1-es optimalizáció kiválasztja a 2-es optimalizáció hiperparamétereinek értékeit. A 2-es optimalizáció 200 alkalommal elvégzi az Ackley függvény optimalizációját. Tekintve, hogy a tényleges problémánál is négy tervezési változó került meghatározásra. Az Ackley függvény egymástól független futtatásakor talált legjobb függvényértékek eltárolásra kerülnek. Az 1-es optimalizáció célfüggvénye pedig az eltárolt eredmények átlaga plusz a szórásuk háromszorosa. A célfüggvény minimuma $f(\mathbf{x}^*) = 0$.

A szórás háromszoros súlyozása az általam vélt fontosságát tükrözi az átlaggal szemben: minél kisebb ez az érték, annál konzisztensebben talál kielégítő eredményt az optimalizációs algoritmus. Előnyösebbnek tartok egy olyan algoritmust, ami nem találja meg a globális minimumot, csak egy kielégítően jó lokális minimumot, de azt megismételhetően, mint egy olyat, ami futtatások egy jelentős hányadában megtalálja a globális minimumot, de hasonló arányban szolgál nem kielégítő megoldással.

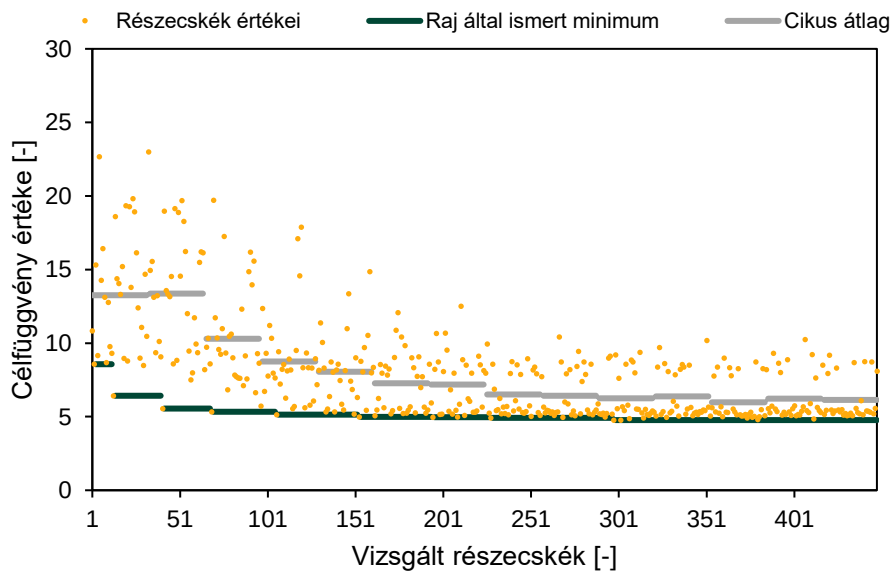
Az algoritmus sztochasztikus jellege miatt nem lehet egyértelműen meghatározni a hiperparaméterek legjobb kombinációját, mivel ugyanazon hiperparaméter értékek mellett minden futtatás más eredményhez vezethet. Ezért szükség van a nagy számú futtatásra, melynek hatására a kiugró jó, vagy rossz futtatások szerepe lecsökken, és az eredményül kapott a hiperparaméter értékek nagyobb eséllyel vezetnek optimum pontra.

4.5.1 Optimalizáció eredménye

Az optimalizáció futásideje 7 óra volt, a legjobb eredmény pedig $f(\mathbf{g}^*) = 4,77$. A $\mathbf{g}^*$ értéke a következő:

$$\mathbf{g}^* = \begin{bmatrix} c_1 \\ c_2 \\ n_r \\ \kappa \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \\ 22 \\ 0,35 \end{bmatrix} \quad (4.12)$$

Az optimalizáció a továbbiakban ezen beállítások mellett futtattam. A kapott eredmény minőségét több szakirodalom is alátámasztja: a c_1 és c_2 értékeit jellemzően a $[0,5; 2,5]$ intervallumon belül szokták megválasztani, leggyakrabban $c_1 = c_2 = 2$ [15], [16], [31]. A futás során vizsgált részecskéket, az ezekhez tartozó célfüggvény értékeket, az optimalizáció által ismert minimumértékeket, valamint az egyes ciklusok eredményeinek átlagát a 4.2. ábra szemlélteti grafikonon.



4.2. ábra Az optimalizáció során kiszámított célfüggvény értékek időrendi sorrendben

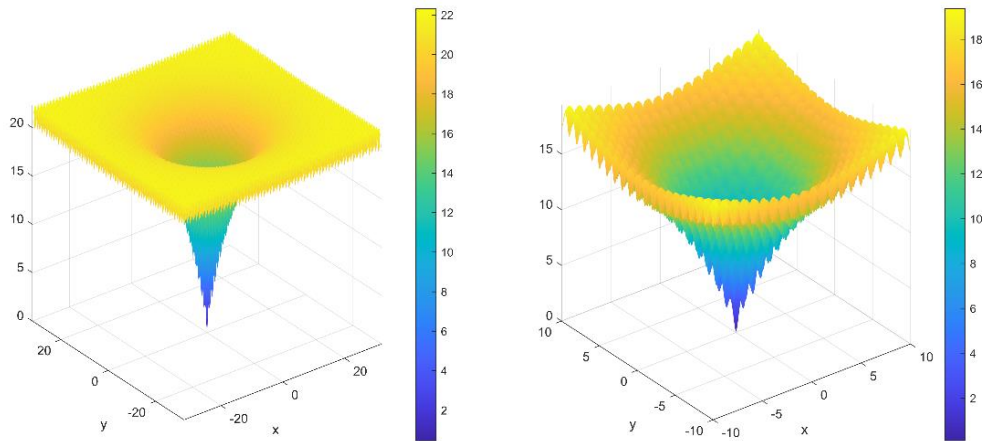
A részecskék mellett szürkével jelölve az egyes ciklusok átlagos értékei láthatók. Megfigyelhető, hogy a részecskék az optimalizáció előrehaladtával egyre inkább „jobb” értékeket vesznek fel. Ez arra enged következtetni, hogy a részecskék lokális minimumot találtak, és ezen pont környezetét fedezték fel. Szintén jól látható a grafikonról, hogy a célfüggvény függvényértékének mozgó átlaga az optimalizáció előrehaladtával egyre inkább konvergál, vagyis minél tovább fut az optimalizáció annál kevesebb lesz az eredmény javulása ugyanannyi számítási teljesítmény ráfordítás mellett.

4.6 Az algoritmus tesztelése

A különböző optimalizációs algoritmusok megbízhatóságát és hatékonyságát gyakran a különböző szakirodalmak által leírt tesztfüggvények segítségével végzik. A tesztfüggvények olyan matematikai függvények, melyek bizonyos tulajdonsága alkalmassá teszi őket optimalizációs algoritmusok tesztelésére. Ez lehet multimodalitás, nem folytonosság, vagy egyéb jellemző [37].

4.6.1 Ackley függvény

$$f(x) = -a \exp \left(-b \sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2} \right) - \exp \left(\frac{1}{n} \sum_{i=1}^n \cos(cx_i) \right) + a + \exp(1) \quad (4.13)$$



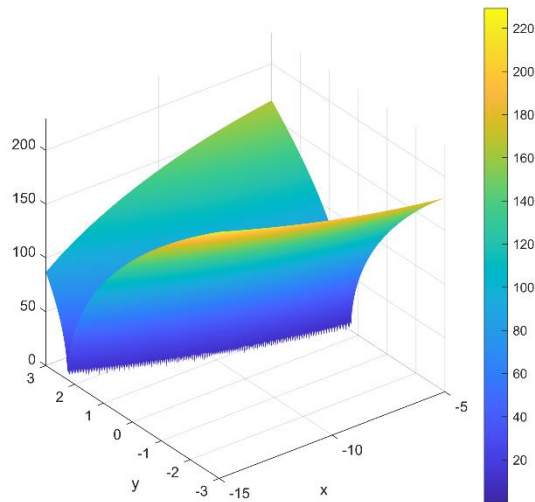
4.3. ábra Az Ackley függvény

A változók javasolt értékei: $a = 20$; $b = 0,2$; $c = 2\pi$. Az értelmezési tartomány általában $x_i \in [-32,768; 32,768]$, de leszűkíthető kisebb intervallumra. A globális minimum $f(\mathbf{x}^*) = 0$, $\mathbf{x}^* = (0; \dots 0)$, $i = 1; 2; \dots n$ [38].

Az Ackley függvényt széles körben alkalmazzák optimalizációs algoritmusok tesztelésére. Kétdimenziós formájában, ahogy a 4.3. ábrán látható, egy közel lapos külső régió jellemzi, és egy nagy mélyedés a középpontban. Fennáll a veszély az optimalizációs algoritmusok – különösen a *hillclimbing* algoritmusok – esetében annak, hogy a számos lokális minimum valamelyikében rekednek [38].

4.6.2 Bukin 6 függvény

$$f(\mathbf{x}) = 100\sqrt{|x_2 - 0,01x_1^2|} + 0,01|x_1 + 10| \quad (4.14)$$

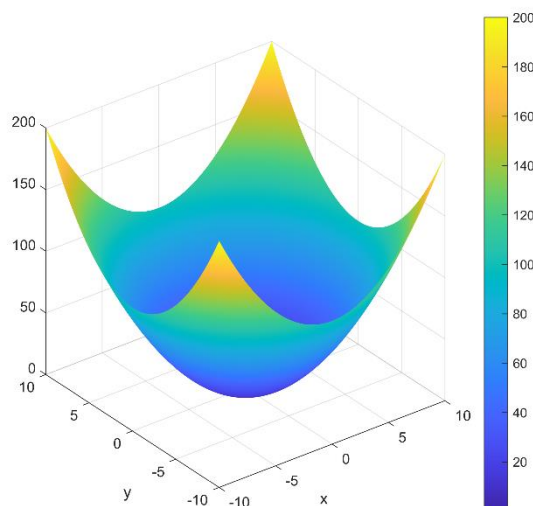


4.4. ábra Bukin 6 függvény

Az értelmezési tartomány általában $x_1 \in [-15; 5]$ és $x_2 \in [-3; 3]$. A globális minimum $f(\mathbf{x}^*) = 0$, $\mathbf{x}^* = (-10; 1)$. Sok lokális minimumponttal rendelkezik, melyek egy völgy mentén helyezkednek el [39].

4.6.3 Gömb függvény

$$f(x) = \sum_{i=1}^n x_i^2 \quad (4.15)$$

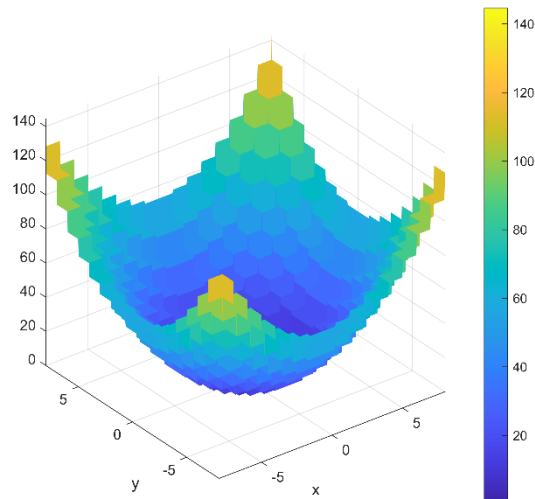


4.5. ábra Gömb függvény

Az értelmezési tartomány általában $x_i \in [-5,12; 5,12]$. A globális minimum $f(x^*) = 0$, $x^* = (0; \dots 0)$, $i = 1; 2; \dots n$ [40]. A függvény csak globális minimummal rendelkezik.

4.6.4 Step függvény

$$f(x) = \sum_{i=1}^n ([x_i] + 0,5)^2 \quad (4.16)$$

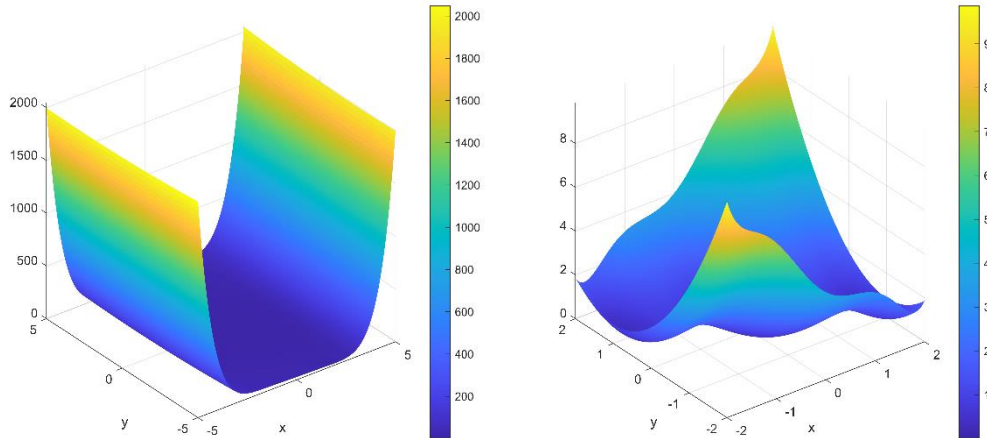


4.6. ábra Step függvény

Az értelmezési tartomány $x_i \in [-100; 100]$. A globális minimum $f(x^*) = 0,5$, $x_i = [0; 1[$, $i = 1; 2; \dots n$. A lapos platók jelenléte nehézséget okoz a gradiens alapú algoritmusoknak [41].

4.6.5 Three-hump camel függvény

$$f(x) = 2x_1^2 - 1,05x_1^4 + \frac{x_1^6}{6} + x_1x_2 + x_2^2 \quad (4.17)$$



4.7. ábra Three-hump camel függvény

A bal oldalon a *three-hump camel* függvény látható az ajánlott korlátok mellett, míg a jobb oldalon a keresési tér egy kis része látható. Erre azért van szükség, hogy a függvény legfontosabb jellemzői megfigyelhetők legyenek. A függvény keresési tere általában $x_1 \in [-5, 5]$, $x_2 \in [-5, 5]$, a globális minimuma pedig $f(x^*) = 0$, $x^* = (0; 0)$. A függvény a globális minimumot is beleértve három lokális minimumponttal rendelkezik [42].

4.6.6 Tesztelés eredménye

A tesztelés megvalósítása a gyakorlatban a következő: Minden tesztfüggvényre nézve lefuttattam az optimalizációt 100 egymástól független alkalommal. Utána kiértékelem ezen futtatások eredményét aszerint, hogy milyen arányban sikerült megtalálni a globális minimumot, valamint, hogy milyen arányban sikerült megtalálni egy előzetesen kielégítőnek választott függvényértéket. Kielégítőnek választom azt a megoldást, ahol a legjobb célfüggvényérték 2 alatti.

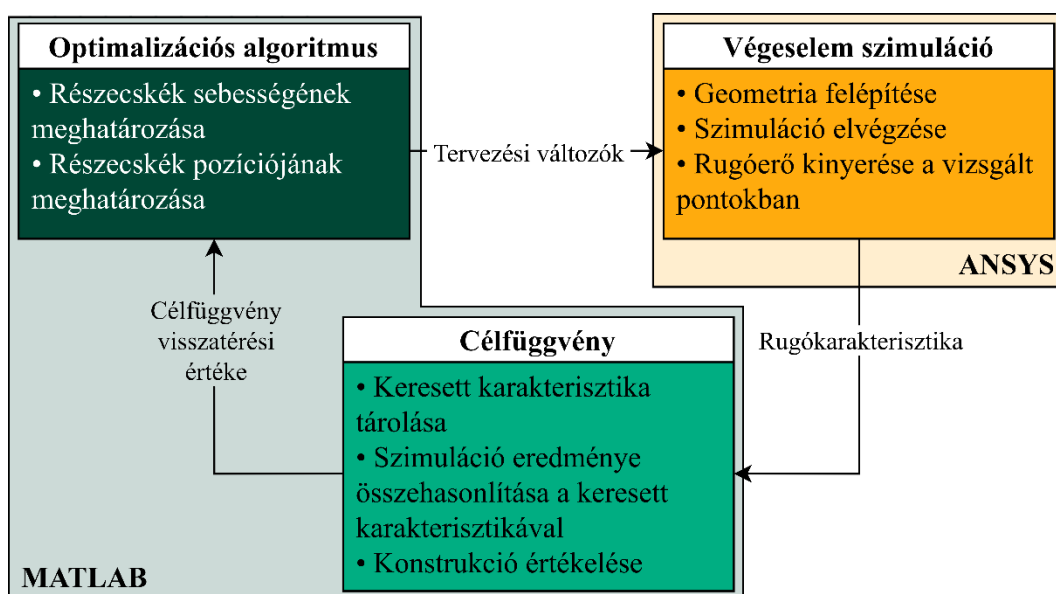
Tesztfüggvény	Globális optimum megtalálásának sikeressége	Kielégítő érték megtalálásának sikeressége
Ackley	34%	84%
Bukin 6	21%	87%
Gömb	32%	98%
Step	37%	91%
Three-hump camel	96%	100%

4.1. táblázat A tesztelések sikeressége

Összességében elmondható, hogy az optimalizációs algoritmus viszonylag kis ciklus- és részecskeszám mellett képes kielégítően jó eredményt találni az esetek túlnyomó többségében. Annak ellenére, hogy nem ismerem a tényleges optimalizációs probléma célfüggvényének jellegét, megfelelőnek tartom az algoritmust az alkalmazásra, tekintve, hogy az általam vizsgált tesztfüggvények kellően változatos tulajdonságokkal bírnak, ideértve a multimodalitást, az elnyúló völgyet és a lapos platókat.

5 Az optimalizáció futtatása

Az optimalizációs algoritmust integrálása a már meglévő tervezési folyamatba akkor a legcélravezetőbb, ha a már használt szoftverek, valamint a már bevett folyamatok cseréje, illetve nagyfokú módosítása nélkül végezhető, ezért az algoritmus megírásakor törekedni kell a modularitásra.



5.1. ábra Az optimalizációs eljárás szoftveres felépítése

Az 5.1. ábrán az optimalizációs folyamat vázlatos felépítése látható, ami három főbb részegységre tagolódik. Az egységek kommunikációja, körfolyamatot eredményez, ami a tradicionális tervezési folyamatot automatizálttá teszi.

5.1 A MATLAB és Ansys közötti kapcsolat megteremtése

Az Ansys Workbench lehetővé teszi a grafikus felületen keresztül végrehajtott műveletek rögzítését, ez a *journaling* (naplózás). Az így keletkezett *journal* fájlok Python-alapú szkriptek formájában kerülnek rögzítésre. Segítségükkel automatizálhatóvá válnak a repetitív feladatok [43].

A MATLAB – Ansys kapcsolat ezen *journal* fájlok segítségével került kialakításra. A végeelem szimuláció felépítését és a tervezési változók parametrizálását követően felvételre került a *journal* szkript. A felvett lépések a következők: a projekt megnyitása, a tervezési változók értékeinek módosítása, a szimuláció futtatása, a kapott mérési pontok exportálása. A kapott *journal* fájl magában tartalmazza a tervezési változók megadott értékeit, ezek változtatásával újbóli futtatáskor új geometria kerül felépítésre és szimulálásra. Ily módon a *journal* fájlban foglalt értékek átírásával az optimalizáció által generált részecske pozíciók átadásra kerülnek a végeelem szimuláció számára.

A módszer előnye, hogy nincs szükség MATLAB specifikus toolboxra, a két szoftver közötti kapcsolat gyakorlatilag egy szövegfájlon keresztül jön létre, így bármely programnyelv használható az optimalizáció megírására és futtatására. Hátránya azonban, hogy *journal* szkriptek segítségével nem lehet irányítani az integrált modulokon belül végzendő műveleteket, ezért tudatosan kell megválasztani parametrizálandó értékeket.

A felvett *journal* fájl sablonként szolgál, a tervezési változók értékei helyett *placeholder* kifejezések kerültek beszúrásra pl.: *var_A*. Ezeket a MATLAB képes megtalálni, és kicserélni az adott részecske koordinátáira. Az 5.2. ábra bal oldalán a sablon fájl részlete látható, színessel kiemelve a *placeholder* kifejezések, míg az ábra jobb oldalán a ténylegesen futtatott *journal* szkript látható, a tervezési változók már behelyettesített értékeivel.

<pre>31 designPoint1 = Parameters.GetDesignPoint(Name="0") 32 parameter1 = Parameters.GetParameter(Name="P1") 33 designPoint1.SetParameterExpression(34 Parameter=parameter1, 35 Expression="var_H [mm]") 36 parameter2 = Parameters.GetParameter(Name="P2") 37 designPoint1.SetParameterExpression(38 Parameter=parameter2, 39 Expression="var_R [mm]") 40 parameter3 = Parameters.GetParameter(Name="P3") 41 designPoint1.SetParameterExpression(42 Parameter=parameter3, 43 Expression="var_A [degree]") 44 parameter4 = Parameters.GetParameter(Name="P4") 45 designPoint1.SetParameterExpression(46 Parameter=parameter4, 47 Expression="var_r [mm]")</pre>	<pre>31 designPoint1 = Parameters.GetDesignPoint(Name="0") 32 parameter1 = Parameters.GetParameter(Name="P1") 33 designPoint1.SetParameterExpression(34 Parameter=parameter1, 35 Expression="84 [mm]") 36 parameter2 = Parameters.GetParameter(Name="P2") 37 designPoint1.SetParameterExpression(38 Parameter=parameter2, 39 Expression="65 [mm]") 40 parameter3 = Parameters.GetParameter(Name="P3") 41 designPoint1.SetParameterExpression(42 Parameter=parameter3, 43 Expression="141 [degree]") 44 parameter4 = Parameters.GetParameter(Name="P4") 45 designPoint1.SetParameterExpression(46 Parameter=parameter4, 47 Expression="44.5 [mm]")</pre>
--	---

5.2. ábra A *journal* sablon, illetve a futtatható szkript

Fontos szem előtt tartani, hogy a MATLAB és az Ansys által készített *journal* fájl is pontot használ tizedesvessző helyett, annak ellenére, hogy az Ansys GUI (*Graphical user interface*) környezetében vessző jelenik meg magyar nyelvi beállítások mellett. A *placeholder* kifejezések cseréjéért az 5.1. forráskód felel:

```
fid = fopen('journalPath/journal_template.wbjn', 'r');  
f = fread(fid, '*char')';  
fclose(fid);  
f = strrep(f, 'var_A' , num2str(var_A));  
f = strrep(f, 'var_R' , num2str(var_R));  
f = strrep(f, 'var_r' , num2str(var_r));  
f = strrep(f, 'var_H' , num2str(var_H));  
fid = fopen('journalPath/journal.wbjn', 'w');  
fprintf(fid, '%s', f);  
fclose(fid);
```

5.1. forráskód A *placeholder* kifejezések cseréjéért a következő kódrészlet

A szimuláció indítását a következő MATLAB parancs végzi:

```
system('"ansysPath/v212/Framework/bin/Win64/RunWB2.exe"  
-X -R journalPath/journal.wbjn');
```

5.2. forráskód A szimuláció indításáért felelős kódrészlet

ahol az *ansysPath* az Ansys elérési útja, míg a *journalPath* a ténylegesen futtatandó *journal* szkript elérési útja. Az *-X* parancssori argumentum felel azért, hogy az Ansys Workbench GUI ablakban induljon, valamint a szkript futása után kerüljön bezárásra. A Workbench futtatható *batch* módban is *-X* helyett *-B* argumentumot használva. Batch módban nem jelenik meg a GUI, a futtatás *console* ablak segítségével történik. Az *-R* pedig azért felel, hogy a Workbench indulásakor a megadott szkript kerüljön futtatásra [43].

A szimuláció futtatásával kapott vizsgálati pontok .csv fájlként kerülnek exportálásra. A MATLAB az ebből kinyert adatokkal kiszámolja a célfüggvény visszatérési értékét, és folytatódik az optimalizáció.

5.2 A futtatás és az eredmények kiértékelése

Az optimalizáció futtatásához ki kell jelölni egy optimális konstrukciót, amit elfogadunk keresett rugókarakterisztikának. Ezen konstrukció tetszőlegesen került kiválasztása annak érdekében, hogy az algoritmust tesztelni lehessen. Legyen az előzetesen választott globális optimum:

$$\mathbf{x}^* = \begin{bmatrix} D \\ d \\ H \\ \alpha \end{bmatrix} = \begin{bmatrix} 114 \text{ mm} \\ 87 \text{ mm} \\ 77 \text{ mm} \\ 142^\circ \end{bmatrix} \quad (5.1)$$

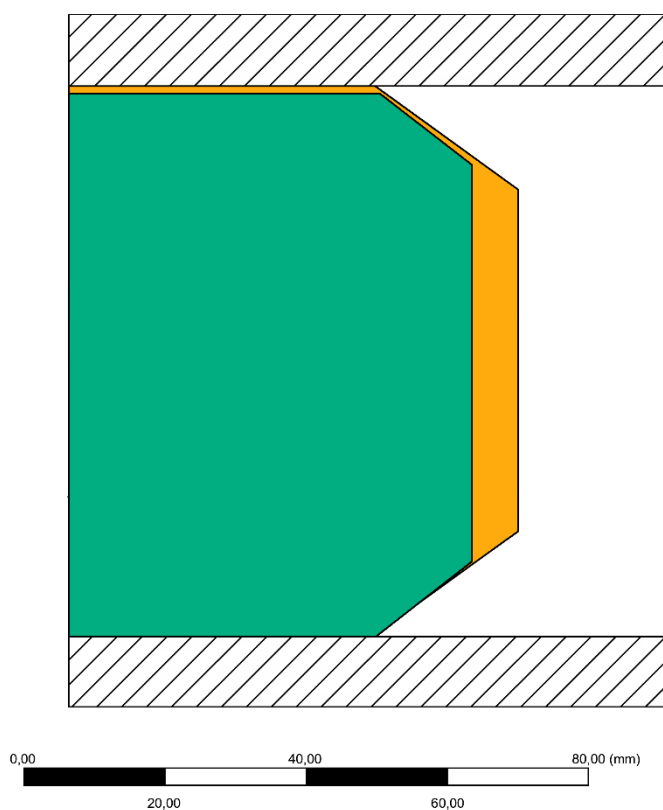
A szimuláció ezekkel a tervezési változókkal került futtatásra, az így kapott **D** keresett rugókarakterisztika pedig el lett tárolva annak érdekében, hogy össze

lehessen hasonlítani az optimalizáció során kapott $\mathcal{S}$ szimulált rugókarakterisztikákkal.

Az optimalizáció futásideje 221 perc volt, gyorsabb, mint a várt futás. Az optimalizáció során talált legjobb célfüggvény érték $f(\mathbf{g}^*) = 0,002651$, $\mathbf{g}^*$ pedig a következő:

$$\mathbf{g}^* = \begin{bmatrix} D \\ d \\ H \\ \alpha \end{bmatrix} = \begin{bmatrix} 127 \text{ mm} \\ 86 \text{ mm} \\ 78 \text{ mm} \\ 144^\circ \end{bmatrix} \quad (5.2)$$

Az optimálisnak kijelölt-, illetve az optimalizáció során talált legjobb rugó geometriájának viszonyát az 5.3. ábra szemlélteti, ahol az optimális rugó zölddel, míg a talált legjobb rugó sárgával van ábrázolva.



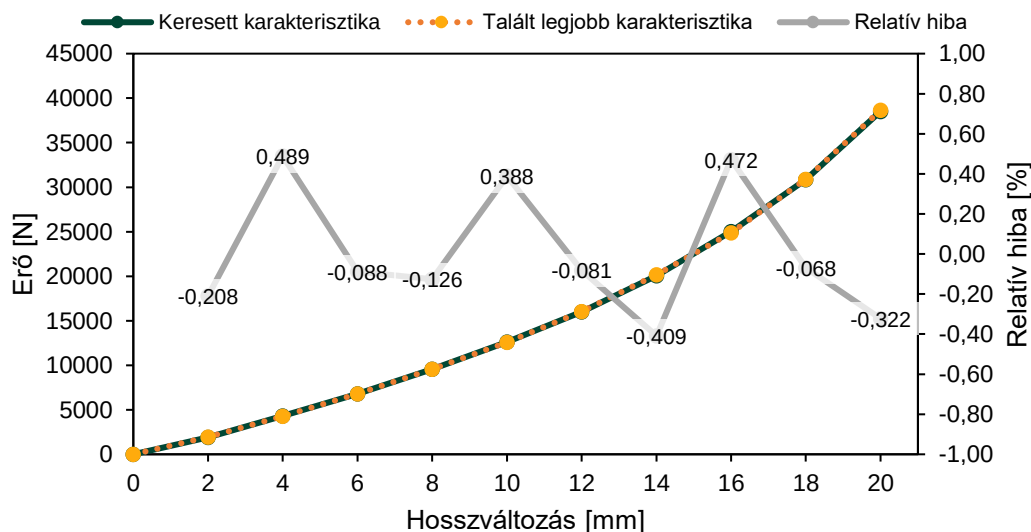
5.3. ábra A keresett-, illetve a talált legjobb rugó geometriája

Az 5.1. táblázat összehasonlítja a **D** és **S** rugókarakterisztikát a vizsgálati pontok segítségével.

Vizsgálati pont [-]	Δl [mm]	Keresett rugókarakterisztika [N]	Talált legjobb rugókarakterisztika [N]	Relatív hiba [%]
1	2	1919	1923	-0,208
2	4	4294	4273	0,489
3	6	6801	6807	-0,088
4	8	9558	9570	-0,126
5	10	12613	12564	0,388
6	12	16003	16016	-0,081
7	14	20071	20153	-0,409
8	16	25011	24893	0,472
9	18	30848	30869	-0,068
10	20	38521	38645	-0,322

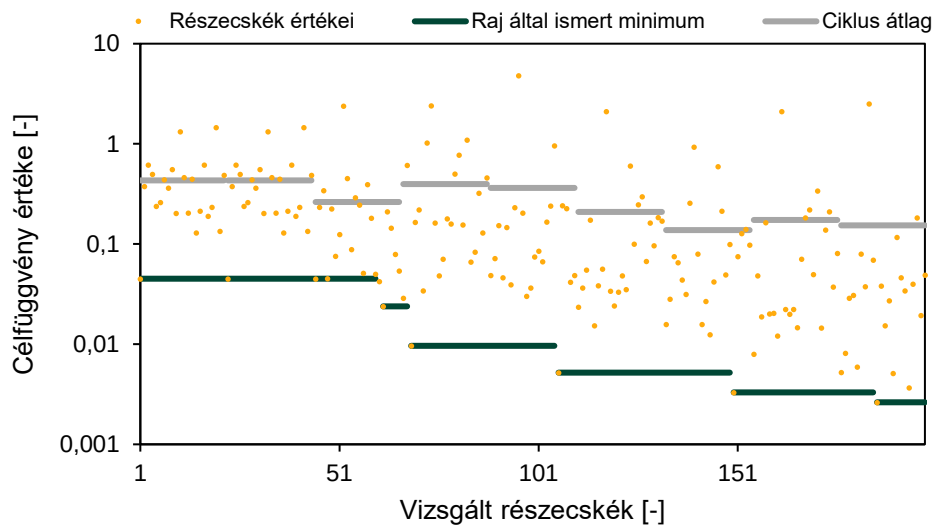
5.1. táblázat A keresett- és az optimalizáció során talált legjobb rugókarakterisztika összehasonlítása

Az optimalizációnak sikerült egy globális optimumhoz közeli, kielégítően jó megoldást találnia. Az egyes vizsgálati pontok közötti eltérések nagyságrendileg elenyésző, a rugókarakterisztika megegyezik, a két görbe – ahogy a 5.4. ábrán is látható – gyakorlatilag fedi egymást. A relatív hiba $-0,5$ és $+0,5$ % között van.



5.4. ábra A keresett-, illetve a talált legjobb rugókarakterisztika

Az 5.5. ábra az optimalizáció során vizsgált részecskéket, az ezekhez tartozó célfüggvény értékeket, az optimalizáció által ismert minimumértékeket, valamint az egyes ciklusok eredményeinek átlagát szemlélteti.



5.5. ábra Az optimalizáció során kiszámított célfüggvény értékek időrendi sorrendben

Összességében az optimalizációt sikeresnek tartom. Annak ellenére, hogy az algoritmus nem találta meg az optimális konstrukciót, mégis képes volt egy optimum-közel rendkívül hasonló rugókarakterisztikával rendelkező konstrukciót találni, mely eleget tesz a működési feltételeknek. A kifejlesztett módszer pedig jól használható, robosztus, valamint könnyen adaptálható számos problémára, így ipari alkalmazásra megfelel, valamint további kutatás témáját is képezheti az eljárás tesztelése és fejlesztése. Továbbfejlesztési lehetőség lehet a hiperparaméterek helyettesítő függvénnyel való beállítása, az optimalizáció futása során a c_1 , c_2 együttthatók lineáris interpolálása, az egyes részecskék értékének párhuzamos számítása, valamint a részecske-raj optimalizáció kombinálása lokális keresőeljárással.

6 Összefoglalás

A szakirodalmi áttekintésem során az optimalizációs fogalmakkal és az optimalizációs algoritmusokkal foglalkoztam. A legfontosabb előrelépés ezen a területen az elektronikus számítógépek megjelenése volt. A megoldható problémák komplexitása a számítógépek számítási teljesítményével együtt nőtt. Az ennek eredményeként kifejlesztett metaheurisztikus módszerek jellemzően magas szintű, iteratív keresőeljárások, melyek segítségével viszonylag gyorsan és hatékonyan lehet optimum-közelí eredményt találni. Gyakran alkalmazzák a globális- és lokális optimumkeresés valamilyen kombinációját annak érdekében, hogy effektíven fel tudja fedezni a keresési teret, mindeközben elkerülve a lokális minimumba való ragadást. Ezek a módszerek kifejezetten effektívnek bizonyultak, így előszeretettel használják komplex problémák megoldására.

A gyakorlati optimalizációs probléma célja egy gumirugó tervezési változóinak megválasztása oly módon, hogy a rugókarakterisztikája minél inkább megközelítse az előre optimálisnak megválasztott karakterisztikát. A különböző rugókarakterisztikákat végeelem szimuláció segítségével nyertem ki. A probléma megoldásához részecske-raj optimalizációt használtam, ami egy rendkívül robusztus és effektív metaheurisztikus eljárás, számos problémára alkalmazható minimális módosítással. Az algoritmust teszteltem, a hiperparamétereit pedig optimalizáltam azért, hogy minél nagyobb eséllyel találjon kielégítően jó konstrukciót. A végeelem optimalizáció felépítése után létrehoztam a kapcsolatot a MATLAB és az Ansys Workbench között, majd futtattam az optimalizációt.

Annak ellenére, hogy az algoritmus nem tudta megtalálni az előzetesen optimálisnak kijelölt konstrukciót, az optimalizáció mégis sikeresnek tekinthető, mivel viszonylag kis számítási teljesítmény mellett is képes volt egy rendkívül hasonló karakterisztikával rendelkező konstrukciót találni. Az optimális- és az optimalizáció során talált legjobb karakterisztika vizsgálati pontjaiban számított relatív hiba $-0,5\%$ és $+0,5\%$ között mozog, így a működési követelményeket teljesíti.

Irodalomjegyzék

- [1] J. R. R. A. Martins és A. Ning, „Introduction”, in *Engineering Design Optimization*, 1. kiad., Cambridge University Press, 2021. doi: 10.1017/9781108980647.
- [2] M. J. Kochenderfer és T. A. Wheeler, „Introduction”, in *Algorithms for optimization*, Cambridge, Massachusetts: The MIT Press, 2019.
- [3] X.-S. Yang, „A Brief History of Optimization”, in *Engineering optimization: an introduction with metaheuristic applications*, Hoboken, N.J: John Wiley, 2010.
- [4] D. H. Ackley, „Analytic properties”, in *A Connectionist Machine for Genetic Hillclimbing*, köt. 28, in The Kluwer International Series in Engineering and Computer Science, vol. 28., Boston, MA: Springer US, 1987. doi: 10.1007/978-1-4613-1997-9.
- [5] S. M. Sinha, „Introduction”, in *Mathematical programming: theory and methods*, 1st ed., New Delhi: Elsevier, 2006.
- [6] Körtélyesi G., „Bevezetés”, in *Mérnöki optimalizáció*, Körtélyesi Gábor, Szerk., Typotex Kiadó, 2012.
- [7] C. Blum és A. Roli, „Metaheuristics in combinatorial optimization: Overview and conceptual comparison”, *ACM Comput. Surv.*, köt. 35, sz. 3, o. 268–308, szept. 2003, doi: 10.1145/937503.937505.
- [8] F. Glover és G. A. Kochenberger, Szerk., „Preface”, in *Handbook of metaheuristics*, in International series in operations research & management science, no. 57., Boston: Kluwer Academic Publishers, 2003.
- [9] A. N. Sadigh, H. Mokhtari, M. Iranpoor, és S. M. T. F. Ghomi, „Cardinality Constrained Portfolio Optimization Using a Hybrid Approach Based on Particle Swarm Optimization and Hopfield Neural Network”, *Adv. Sci. Lett.*, köt. 17, sz. 1, o. 11–20, okt. 2012, doi: 10.1166/asl.2012.3666.
- [10] C. Reeves, „Genetic Algorithms”, in *Handbook of metaheuristics*, F. Glover és G. A. Kochenberger, Szerk., in International series in operations research & management science, no. 57., Boston: Kluwer Academic Publishers, 2003.

- [11] Borgulya I., „Biológiai eljárás, viselkedés ösztönözte módszerek”, in *Optimalizálás evolúciós számításokkal*, Budapest: Typotex Elektronikus Kiadó Kft., 2012.
- [12] Gregorics T., „Kereső rendszerek a mesterséges intelligenciában”, in *Mesterséges intelligencia*, Futó Iván, Szerk., Budapest: Aula, 1999.
- [13] A. E. Eiben és J. E. Smith, „What Is an Evolutionary Algorithm?”, in *Introduction to Evolutionary Computing*, 2. kiad., in Natural Computing Series. , Berlin, Heidelberg: Springer Berlin, Heidelberg, 2015. doi: 10.1007/978-3-662-44874-8.
- [14] M. Dorigo és T. Stützle, „The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advance”, in *Handbook of metaheuristics*, F. Glover és G. A. Kochenberger, Szerk., in International series in operations research & management science, no. 57., Boston: Kluwer Academic Publishers, 2003.
- [15] J. Kennedy és R. Eberhart, „Particle swarm optimization”, in *Proceedings of ICNN'95 - International Conference on Neural Networks*, Perth, WA, Australia: IEEE, 1995, o. 1942–1948. doi: 10.1109/ICNN.1995.488968.
- [16] X.-S. Yang, „Particle Swarm Optimization”, in *Nature-inspired optimization algorithms*, First edition., Amsterdam: Elsevier, 2014.
- [17] G. Beni és J. Wang, „Swarm Intelligence in Cellular Robotic Systems”, in *Robots and Biological Systems: Towards a New Bionics?*, P. Dario, G. Sandini, és P. Aebischer, Szerk., Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, o. 703–712. doi: 10.1007/978-3-642-58069-7_38.
- [18] F. E. Fernandes Junior és G. G. Yen, „Particle swarm optimization of deep neural networks architectures for image classification”, *Swarm Evol. Comput.*, köt. 49, o. 62–74, szept. 2019, doi: 10.1016/j.swevo.2019.05.010.
- [19] R. Poli, J. Kennedy, és T. Blackwell, „Particle swarm optimization: An overview”, *Swarm Intell.*, köt. 1, sz. 1, o. 33–57, okt. 2007, doi: 10.1007/s11721-007-0002-0.
- [20] „Rugalmas kötések: rugók”, in *Gépelemek I.*, Budapest: Tankönyvkiadó, 1989.
- [21] Borbás L., „Rugalmas kötések (rugók)”, in *Jármű- és hajtáselemek I.*, Devecz János, Szerk., Typotex Kiadó, 2012.

- [22] „APSOvib® Rubber hollow spring type TO”. Elérés: 2023. augusztus 1. [Online]. Elérhető: <https://www.apsoparts.com/en-INT/12210202-apsovibr-rubber-hollow-spring-type-to.html>
- [23] „Rubber — Tolerances for products — Part 1: Dimensional tolerances”. 2014. július.
- [24] „The forecaster’s toolbox”, in *Forecasting: principles and practice*, 2nd edition., Lexington, Ky.: Otexts, online, open-access textbook, 2018.
- [25] „IBM Documentation”. Elérés: 2023. június 8. [Online]. Elérhető: <https://www.ibm.com/docs/hu/cognos-analytics/11.1.0?topic=forecasting-statistical-details>
- [26] W. Wang és Y. Lu, „Analysis of the Mean Absolute Error (MAE) and the Root Mean Square Error (RMSE) in Assessing Rounding Model”, *IOP Conf. Ser. Mater. Sci. Eng.*, köt. 324, o. 012049, márc. 2018, doi: 10.1088/1757-899X/324/1/012049.
- [27] EPM Information Development Team, „MAPE (Mean Absolute Percentage Error)”, Oracle Help Center. Elérés: 2023. június 11. [Online]. Elérhető: https://docs.oracle.com/en/cloud/saas/planning-budgeting-cloud/pfusu/insights_metrics_MAPE.html
- [28] Huri D., „Végeselemes rendszerhez kapcsolt optimalizáló eljárás fejlesztése gumiütköző tervezéshez”, Debreceni Egyetem, Debrecen, 2022.
- [29] M. A. Cruz Gómez, E. A. Gallardo-Hernández, M. Vite Torres, és A. Peña Bautista, „Rubber steel friction in contaminated contacts”, *Wear*, köt. 302, sz. 1–2, o. 1421–1425, ápr. 2013, doi: 10.1016/j.wear.2013.01.087.
- [30] D. H. Ackley, „Preface”, in *A Connectionist Machine for Genetic Hillclimbing*, köt. 28, in The Kluwer International Series in Engineering and Computer Science, vol. 28., Boston, MA: Springer US, 1987. doi: 10.1007/978-1-4613-1997-9.
- [31] A. P. Engelbrecht, „Particle Swarm Optimization”, in *Computational Intelligence: An Introduction*, 2nd ed., Chichester: John Wiley & Sons, 2007.
- [32] O. Olorunda és A. P. Engelbrecht, „Measuring exploration/exploitation in particle swarms using swarm diversity”, in *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, Hong Kong, China: IEEE, jún. 2008, o. 1128–1134. doi: 10.1109/CEC.2008.4630938.

- [33] Balogh S., „Többszempon্তু gazdasági döntéseket segítő genetikusan algoritmus kidolgozása és alkalmazásai”, Kaposvári Egyetem, Kaposvár, 2009.
- [34] Borgulya István, *Optimalizálás evolúciós számításokkal*. Budapest: Typotex Elektronikus Kiadó Kft., 2012.
- [35] J. Barrera, O. Álvarez-Bajo, J. J. Flores, és C. A. Coello Coello, „Limiting the Velocity in the Particle Swarm Optimization Algorithm”, *Comput. Sist.*, köt. 20, sz. 4, o. 635–645, dec. 2016, doi: 10.13053/cys-20-4-2505.
- [36] „Pseudocode Standard”, Computer Science & Software Engineering - Cal Poly. Elérés: 2023. szeptember 15. [Online]. Elérhető: http://users.csc.calpoly.edu/~jdalbey/SWE/pdl_std.html
- [37] M. Jamil, X.-S. Yang, és H.-J. Zepernick, „Test Functions for Global Optimization”, in *Swarm Intelligence and Bio-Inspired Computation*, Elsevier, 2013, o. 193–222. doi: 10.1016/B978-0-12-405163-8.00008-9.
- [38] D. Bingham és S. Surjanovic, „Ackley Function”. Elérés: 2023. április 3. [Online]. Elérhető: <https://www.sfu.ca/~ssurjano/ackley.html>
- [39] D. Bingham és S. Surjanovic, „Bukin Function N. 6”. Elérés: 2023. április 26. [Online]. Elérhető: <https://www.sfu.ca/~ssurjano/bukin6.html>
- [40] „Sphere Function”. Elérés: 2023. szeptember 23. [Online]. Elérhető: <https://www.sfu.ca/~ssurjano/spheref.html>
- [41] E. P. Adorio, „MVF - Multivariate Test Functions Library in C for Unconstrained Global Optimization”.
- [42] D. Bingham és S. Surjanovic, „Three-Hump Camel Function”. Elérés: 2023. szeptember 6. [Online]. Elérhető: <https://www.sfu.ca/~ssurjano/camel3.html>
- [43] ANSYS, Inc., „Ansys Workbench Scripting Guide”. 2021.