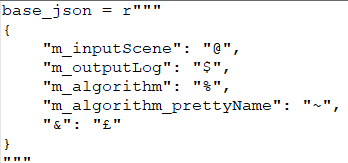
## Implementation av hjälpskript

Utöver Broadmark och simulationsgeneratorn tillhandahöll Serpa & Rodrigues(2019c) även hjälpskripten *run* och *plot*, vilka båda är python skript. Serpa & Rodrigues(2019c) redogör att runär ett skript som automatiserar testningen genom att generera de testfiler, vars struktur kan ses i **figur 19**, som Broadmark använder. Testfilerna innehåller vägbeskrivning till testscenarion samt de algoritmen som ska testas mot det, ett sådant test kommer här efter refereras till som en *testsimulation*. Slutligen tar runtar .json filen som Brodmark producerar, för exempel se **figur 20**, och använder dataanalyserings och editerings verktyget *Pandas* (2008) för att formatera datan från dem på fyra olika sätt, varje formatering sparas ner i en .csv en .pickle samt en .xlsx fill. De resulterande fyra dataformateringarna är:

* *main\_frame*, innehåller tiderna för alla överlappningssökningar för alla testsimulationer
* *main\_described\_frame*, innehåller antalet objekt medelvärdet, standardavvikelsen, maximala, minimala, tiden 25 procent, tiden 50 procent samt tiden 75 procent utav överlappningssökningar för alla testsimulationer.
* *multi\_index\_frame*, innehåller en multindex som grupperar resultatdata utifrån scenariot och där efter utifrån antalet objekt som används.
* *lines\_frame*, innehåller medelvärdet utav tiden överlappnings sökning tog för alla testsimulationer.

Serpa & Rodrigues(2019c) förklarar att *plot* använder datavisualiseringsprogrammet *Seaborn* (2013) för att skapa diagram utifrån den formaterade datan. Diagrammen som skapas är ett linjediagram vilket visar hur prestandan för algoritmerna utvecklas i förhållande till antalet objekt i testscenariona, ett stapeldiagram som redogör för hur algoritmernas medelprestanda för de olika scenariona. Slutligen sammanställer de även ett violindiagram, vilket är en form av låddiagram, för att visa på spridningen utav algoritmernas medelprestanda. Noterbar är att plot endas redogör violindiagrammet för det första testscenariot. Exempel på de diagram *plot* skapar kan ses i **figur 24** till **figur 26**.



1. Formatet på .json filen som Broadmark använder för testningen



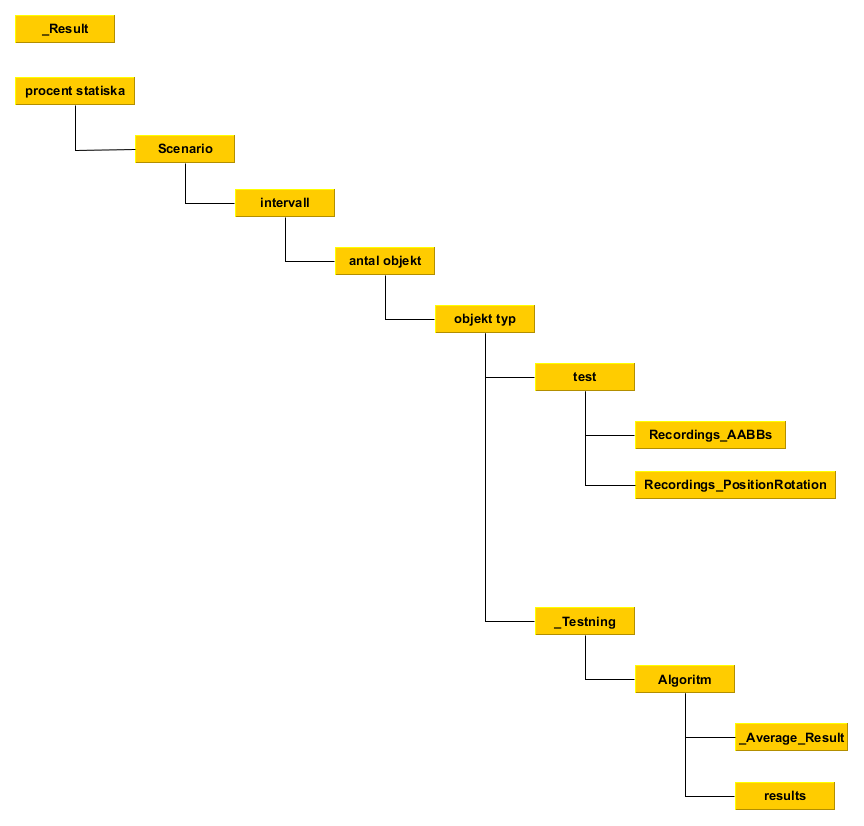
1. Exempel på .json filen innehållandes resultat från en testkörning av CGAL i Broadmarks.

Utifrån deras funktionalitet är run och plot mycket intressanta för denna undersökning. Run skulle underlätta testningen samt så sammanställer det resultatet och tar fram exakt den information som definierades i 3.1.4. Mer specifikt, stapeldiagrammet ger en bra inblick i hur algoritmerna presterar för olika mängder objekt, medan linjediagrammet ger en överblick över algoritmens prestanda utvecklas när antalet objekt ökar. Slutligen violindiagrammet visserligen inte är ett låddiagram men det är besläktat med det och visar samma information ett låddiagram gör.

Dock om run och plot ska användas så kommer det krävas efterforskning kring hur pythons syntax, *Pandas* (2008) samt *Seaborn* (2013) fungerar. Det finns en potentiell risk att de kommer ta för mycket tid att lära sig, om än lite, funktionalitet i två program bibliotek i ett nytt programmeringsspråk. Alternativ är att implementera samma funktionalitet i antingen C# genom att göra ett separat program som utför samma funktionalitet. Alternativt implementera samma funktionalitet med C++ direkt i Broadmark. Oavsett om C# eller C++ skulle väljas kommer det ändå behövas efterforskning för att hitta motsvarande programbiblioteket till *Pandas* (2008) och *Seaborn* (2013) , eller hitta information för att implementera dem från grunden. Utöver det tillkommer även att integrerar det med de redan existerande systemet. Python är dock ett programmeringsspråk vars grundprinciper bygger på lättläslighet, samt är det ett mycket populärt språk så det finns många resurser att hämta information ifrån och lika så är både *Pandas* (2008) och *Seaborn* (2013) mycket väldokumenterade. Det har även fördelen med att de bevisligen fungera i med ramverket och tillhandahåller den önskade funktionarlitet.

Slutsatsen drogs där med att det troligen skulle ta mins lika mycket tid att implementera samma funktionalitet som den som finns i run och plot i C# eller C++ och där utöver måst en sådan läsning även integreras i ramverket. Utifrån det beslutades det där med att vidareutveckla run och plot. För vidareutvecklingen skapades kopior utav run och plot som denoteras EJ\_run och EJ\_plot, det är dessa kopior som vidareutvecklades till att fungera med de expanderade systemet medan run och plot behölls för att ha som referens och för felsökning. Det ska noteras att både run och plot egentligen är uppdelade över flera skript: *plot*, *run*, *bm\_run*, *bm\_parse* och *bm\_plot*. För denna rapport ansågs det dock vara smidigare att endas referera till dem som två enskilda skript. Det kommer även skapas kopior av de övriga skripten, vilka kommer anpassa för att hantera den nya funktionalitet, vilka kommer namnges efter samma standar som kopiorna av run och plot. Dock för enkelhetens skull kommer all funktionalitet som är associerade med run eller plot, även om implementation utav dem är något av de andra skripten, beskrivas som om de är implementerade i dem. Slutligen all python kodediterades med hjälp av *Notepad ++* (2003).

### Körnings skript

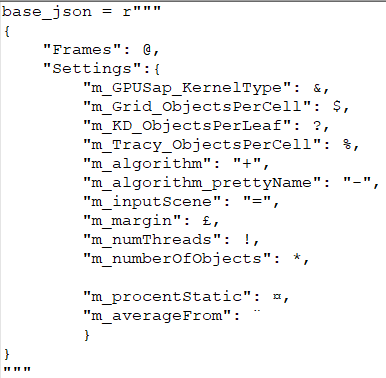


1. Filhierarkin efter att testningen har genomförts.

För att få EJ\_Run att ha den önskade funktionaliteten så kommer grunden i run behöva expanderas för att kunna utnyttja filhierarkin som simulationsgeneratorn skapat. Utöver de kommer dataformatering som run gör behöva modifieras för att ta hänsyn till den nya parametern i form av andelen statiska objekt.

Traverseringe av filhierarkin konstruerades med hjälp utav nästlade loopar, vilka går igenom filhierarkin och testar varje enskild test testscenario. Själva testningen tar även och expanderar filhierarkin i att under objekttyp så adderas en mappen *\_Testning* i vilket innehåller en mapp för varje algoritm som testat. I algoritmerna lagras av upprepningar utav den specifika testsimulationerna samt finns en mapp i vilka resultat från testningen lagarnas. Det finns även en mapp i vilket medelresultat vilket beräknats från alla upprepningar av testscenariot lagras. En kopia av medelresultat lagras även i enseparat \_Resultat mapp, vilken om inget annat anges planerats i hierarkins root, i vilket medelresultat för alla testsimulation lagras. Detta görs för att förenkla EJ\_plot så det inte ska behöva traverserna hela filhierarkin för att komma åt medelresultat. Utseendet av filhierarkin efter att testningen har utförts kan ses i **figur 21**.

Själva beräkningen av medelresultaten beräknades medhjälp av *Pandas* (2008) då de hade funktion som enkelt kunde beräkna medelvärdet får flera datakällor. För medelresultaten så expanderas settingsstrukturen, som kan ses i **figur 20**, för att utöver den redan existerande informationen även innehålla andelen statiska objekt för testet. Samt adderades information från hur många upprepningar som medelresultatet beräknats. Strukturen på den expanderade versionen kan ses i **figur 22**.



1. Expanderade version av settings strukturen som används för medelvärderesultat utav alla upprepningar utav ett specifik testsimulation. m\_procentStatic har adderats för lagra andelen statiska objekt och m\_averageFrom adderas för lagrat från hur många upprepningar som medelvärdet är beräknat ifrån.

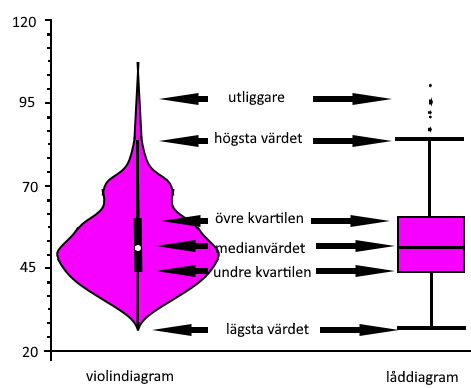
### Resultat visualiserar skriptet

*Seaborn* (2013)

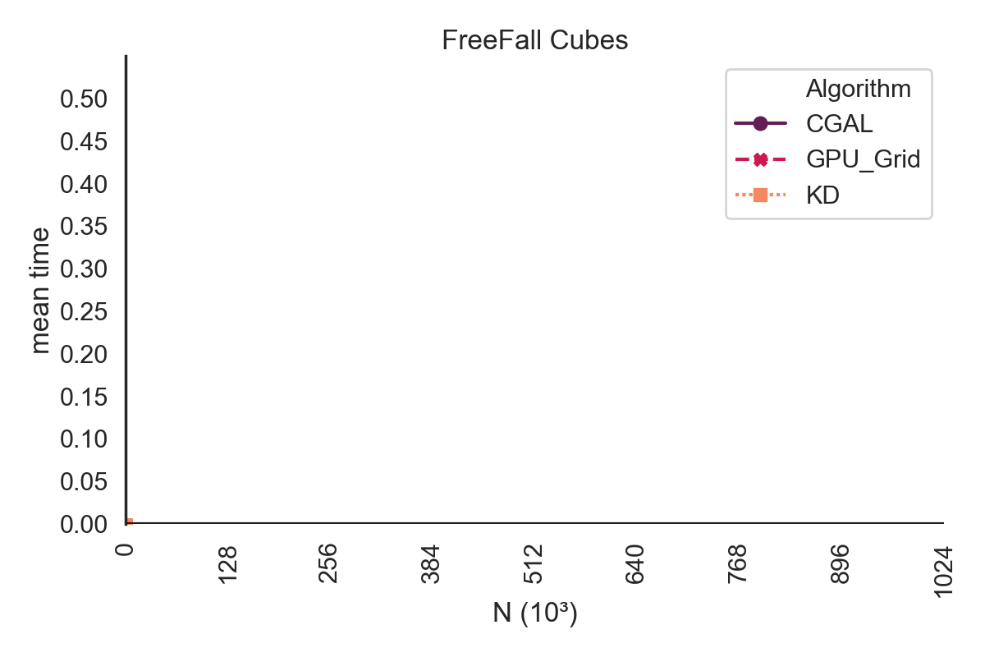
Ej\_plot krävde något mer efterforskning *Seaborn* (2013) och *Pandas* (2008) dock tack vare att de var mycket väldokumenterad var detta inte något som tog mer tid än nödvändigt. Valet gjordes att så långt som möjligt presentera relevanta diagram, exempelvis **figur 28** gjordes med EJ\_plot och vissas alla tester för ett scenario vilket kan jämföras plot som för motsvarande diagram hade presentera varje objekt mängd i individuella diagram som i **figur 25**.

Utifrån tidigare beslut så konverterades violindiagrammet till ett låddiagram, se **figur 30** och **figur 31** för exempel. Vilket var enkelt då *Seaborn* (2013) även hade funktionalitet för att göra låddiagram, samt så hade låddiagram funktionen samma parameter violindiagrammet vilket gjorde modifieringen mycket simpel att implementera. Efter att lite efterforskning kring vad ett violindiagram faktisk var beslutade det dock att återgå till att använda det. Det visade sig nämligen att violindiagram är en kombination av låddiagram och *kernel density estimation*, violindiagrammet visar där med samma data som ett låddiagram gör plus att den använder en täthetsfunktion för att redogöra för sannolikhet för de olika värden, se **figur 23** Hintzer, J., L. & Nelson, R., D.(1998, s.181). för att se skillnaden mellan jämför **figur 30** och **figur 31** med **figur 32** och **figur 33**

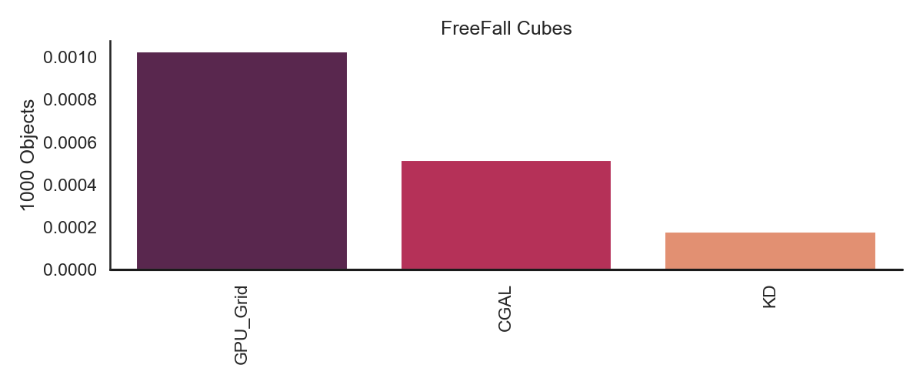
När implementation var klar gjordes en testkörning



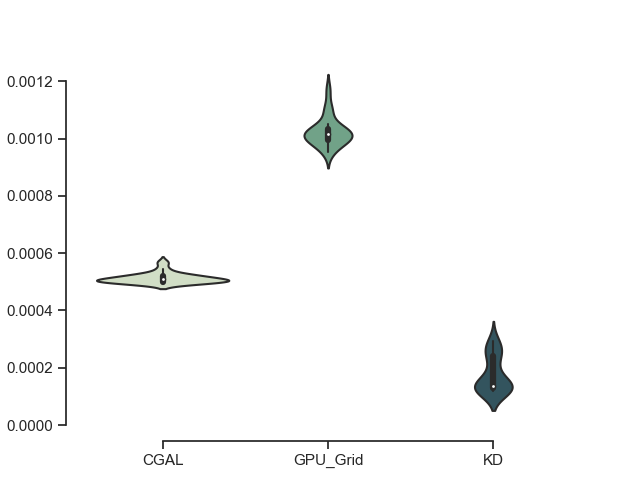
1. Exempel på det de gemensamma delarna för ett låddiagram och ett violindiagram. Det som särskiljer violindiagrammet är att ju bredare fältet är ju större är sannolikheten att ett slumpvist valt värde ur dess datamängd har det värdet.



1. Exempel på linjediagram *plot* producerar, det visar medelvärdet för algoritmernas exekverings tid beroende på storleken på indata. Linjerna går inte att direkt se eftersom endast ett test användes vilket innehöll 1000 objekt.



1. Exempel på stapeldiagrammet *plot* producerar, det Y-axeln visar medel tiden, X-axeln är algoritmen. Antalet objekt som användes i scenariot står på vänster sida och namnet på scenariot och objekten som användes står överst.



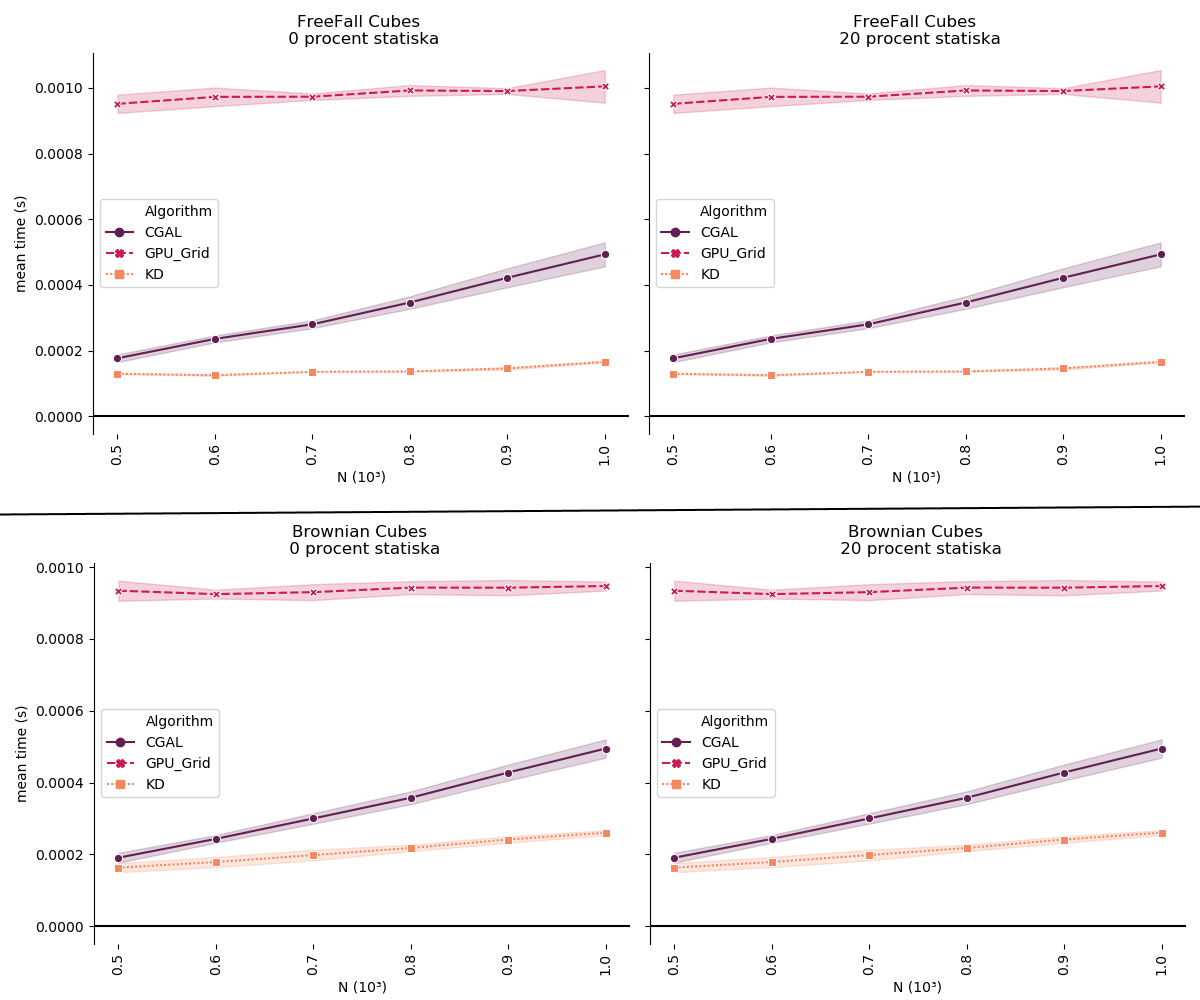
1. Exempel på violndiagram som *plot* producerar, det Y-axeln visar medel tiden, X-axeln är algoritmen.

### Resultat utav testkörning utav EJ\_run och EJ\_plot

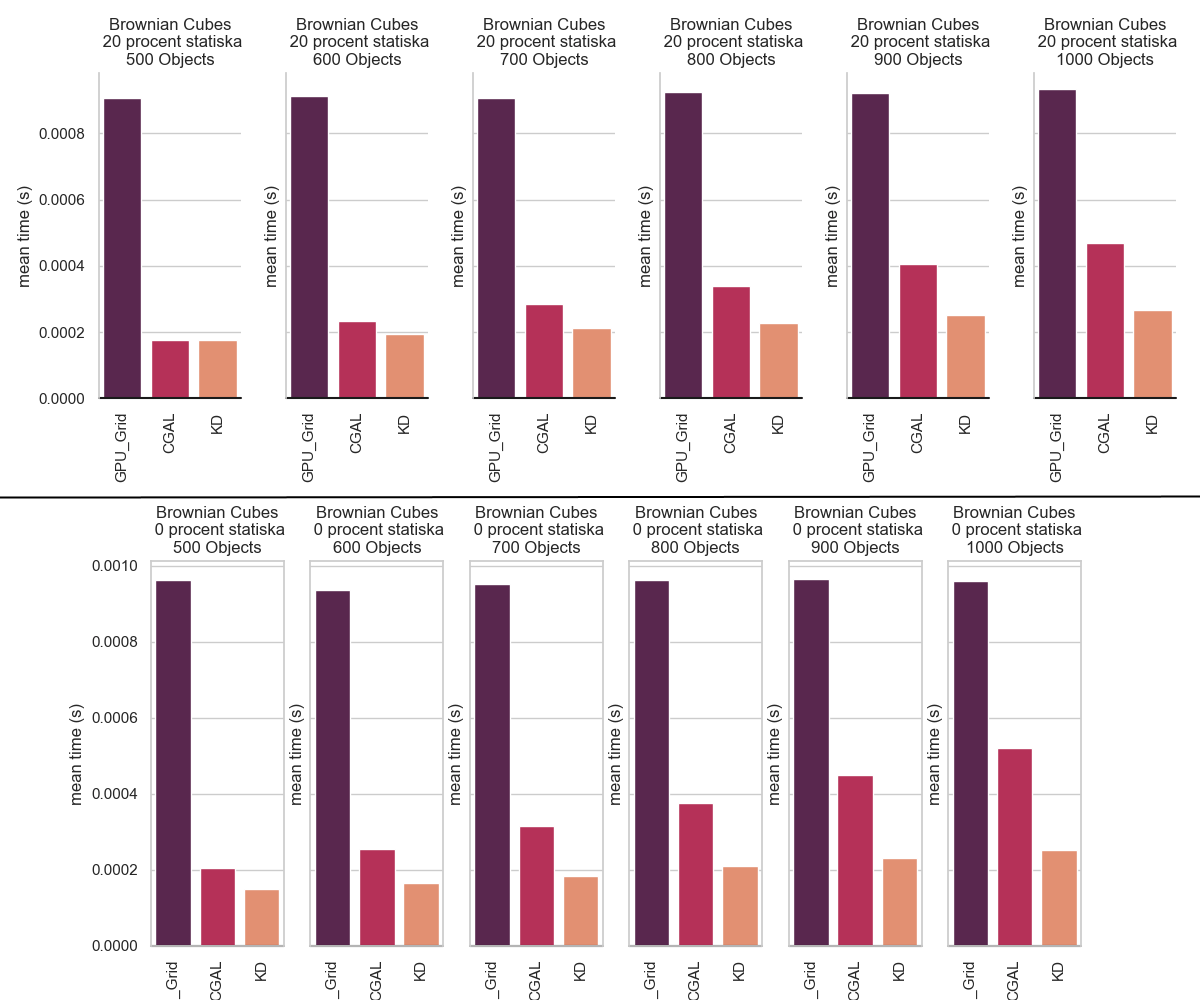
När implementation av EJ\_run och Ej\_plot var klar gjordes en testkörning med följande testserie:

* Scenarion: slumpvandring, fritt fall
* Andel statiska objekt: 0 procent och 20 procnet
* Objekt typ: kuber
* Andel objek: 500, 600, 700, 800, 900, 1000
* Algoritmer: CGAL, KD, grafikprocessor- rutnät

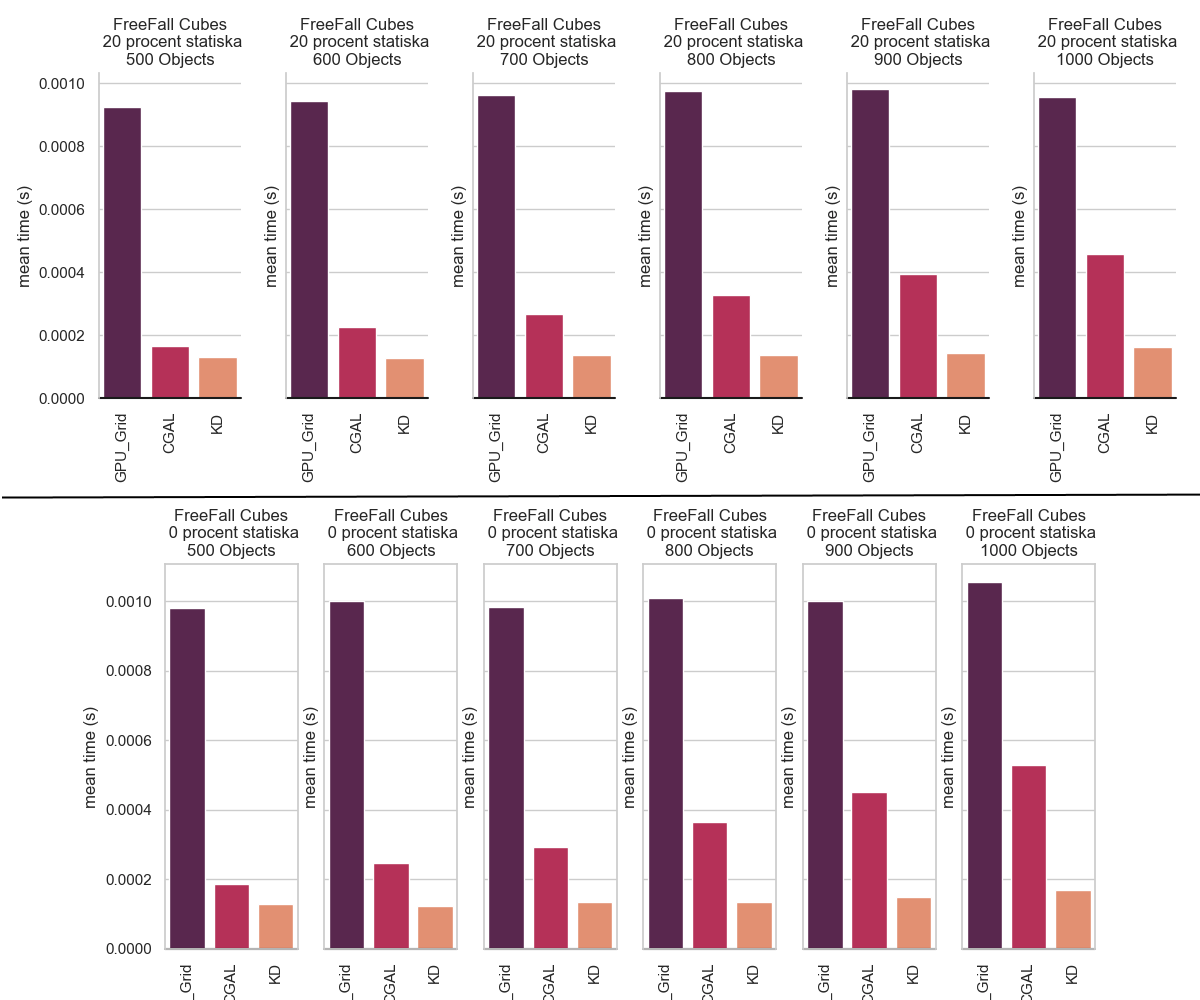
Sedan användes EJ\_plot för att producera diagramen som visas i **figur 27** till och med **figur 34**. **Figur 31** och **figur 32** är som tidigaren nämnt från innan det beslutades att återgå till violindiagrammet som går att se i **figur 32** och **figur 33**



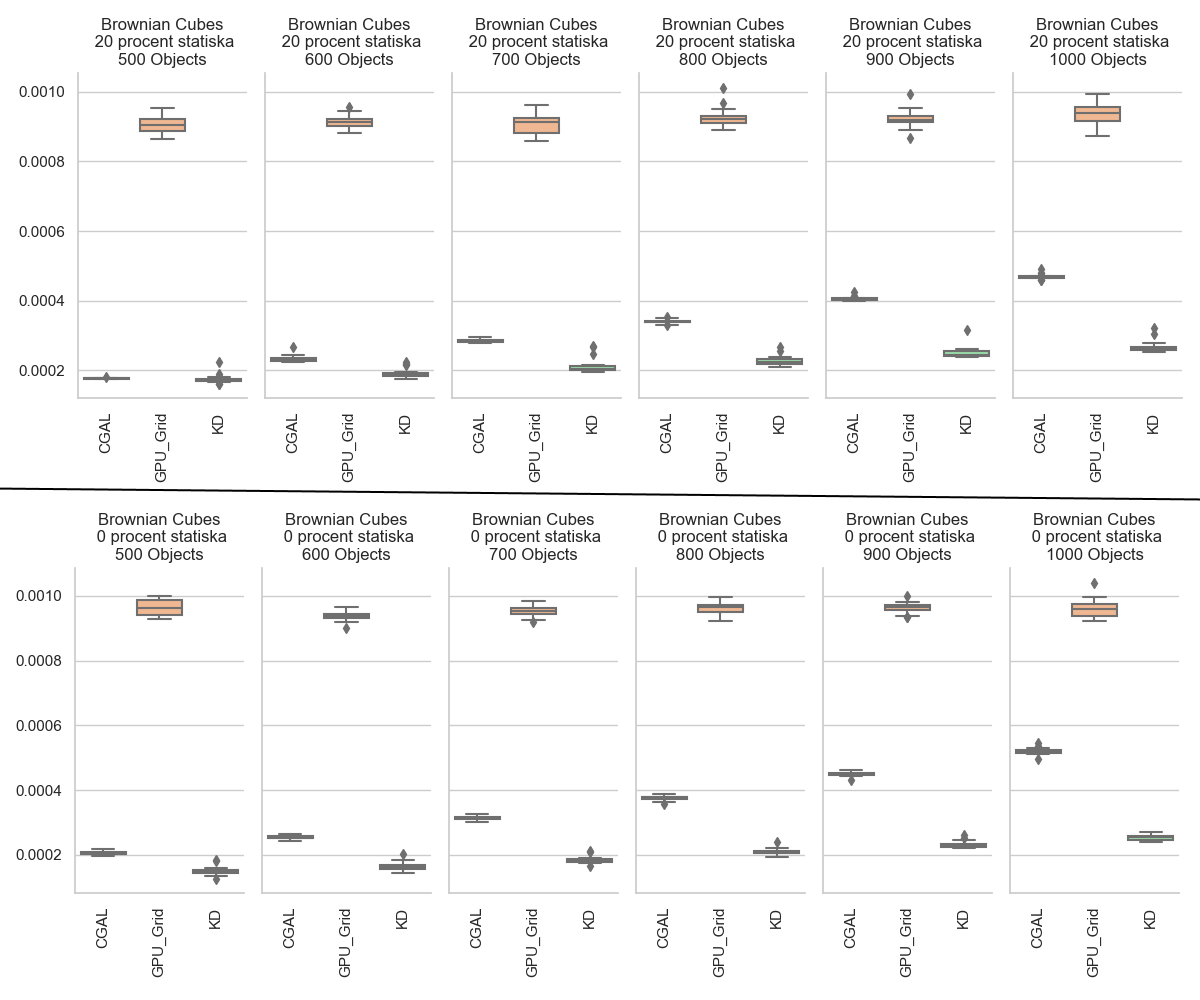
1. Exempel på linjediagram som skapats med EJ\_plot, det visualiserar tre algoritmernas prestanda i förhållande till antalet objetk för två olika scenarion samt två olika förhållanden mellan statiska och dynamiska objekt.

rhållanden mellan dynamiska och feom statiska objekt 

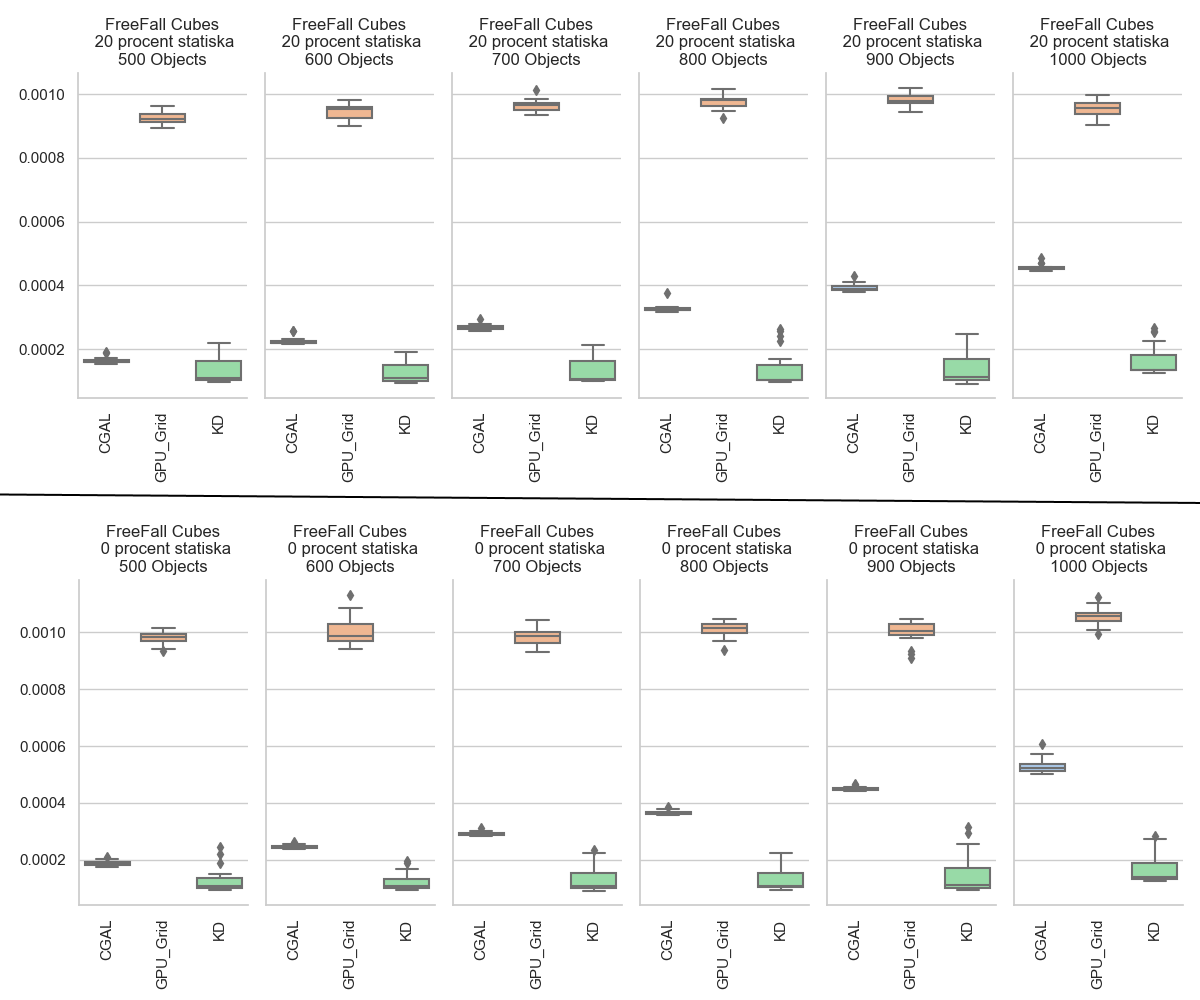
1. Exempel på stapeldiagram från EJ\_plot, resultatet visar medelvärdet för tre algoritmer prestanda för olika mängder objekt i testscenariot slumpvandring med två olika förhållanden mellan statiska och dynamiska objekt.



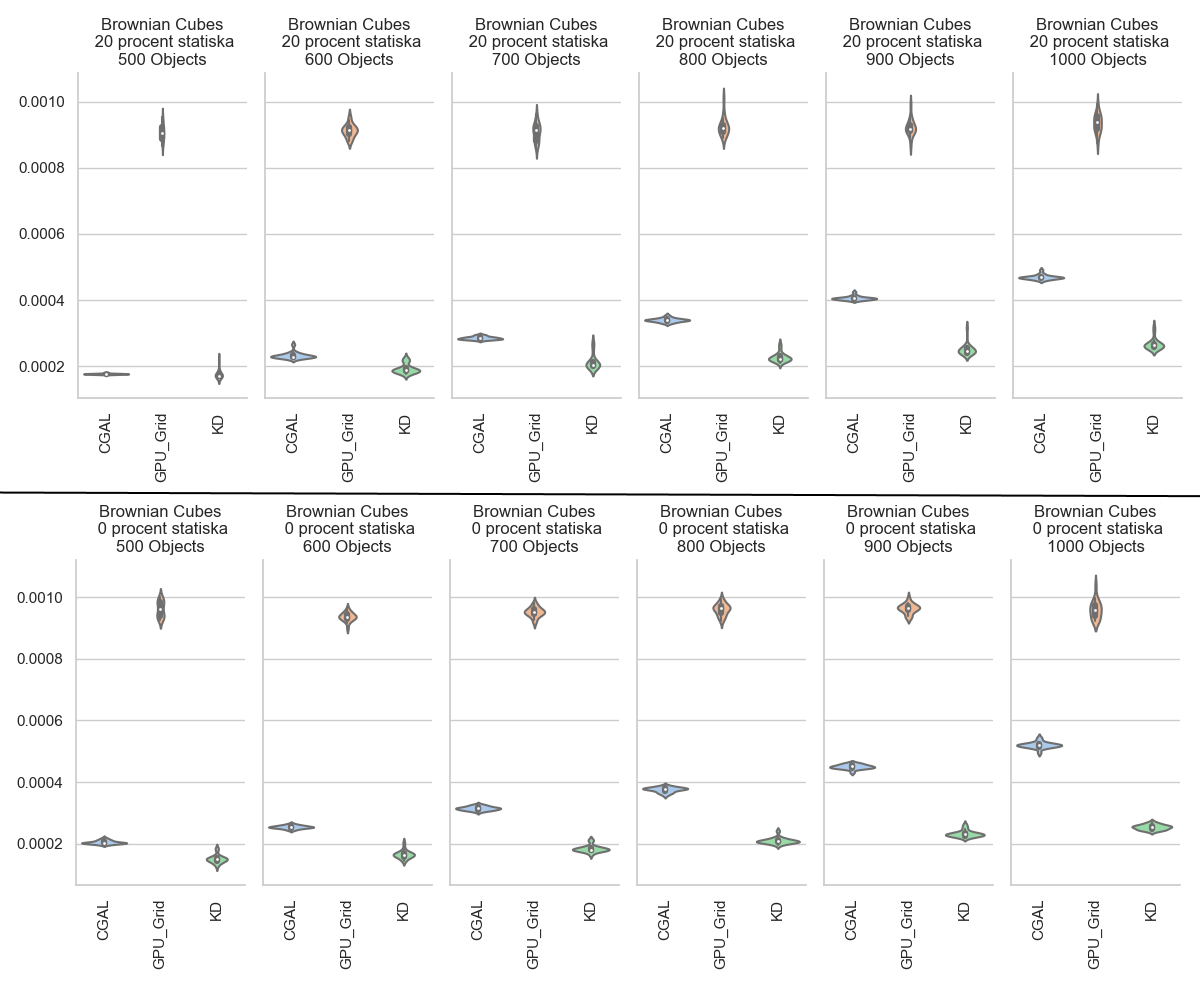
1. Exempel på stapeldiagram från EJ\_plot, resultatet visar medelvärdet för tre algoritmer prestandan för olika mängder objekt i testscenariot fritt fall med två olika förhållanden mellan statiska och dynamiska objekt.



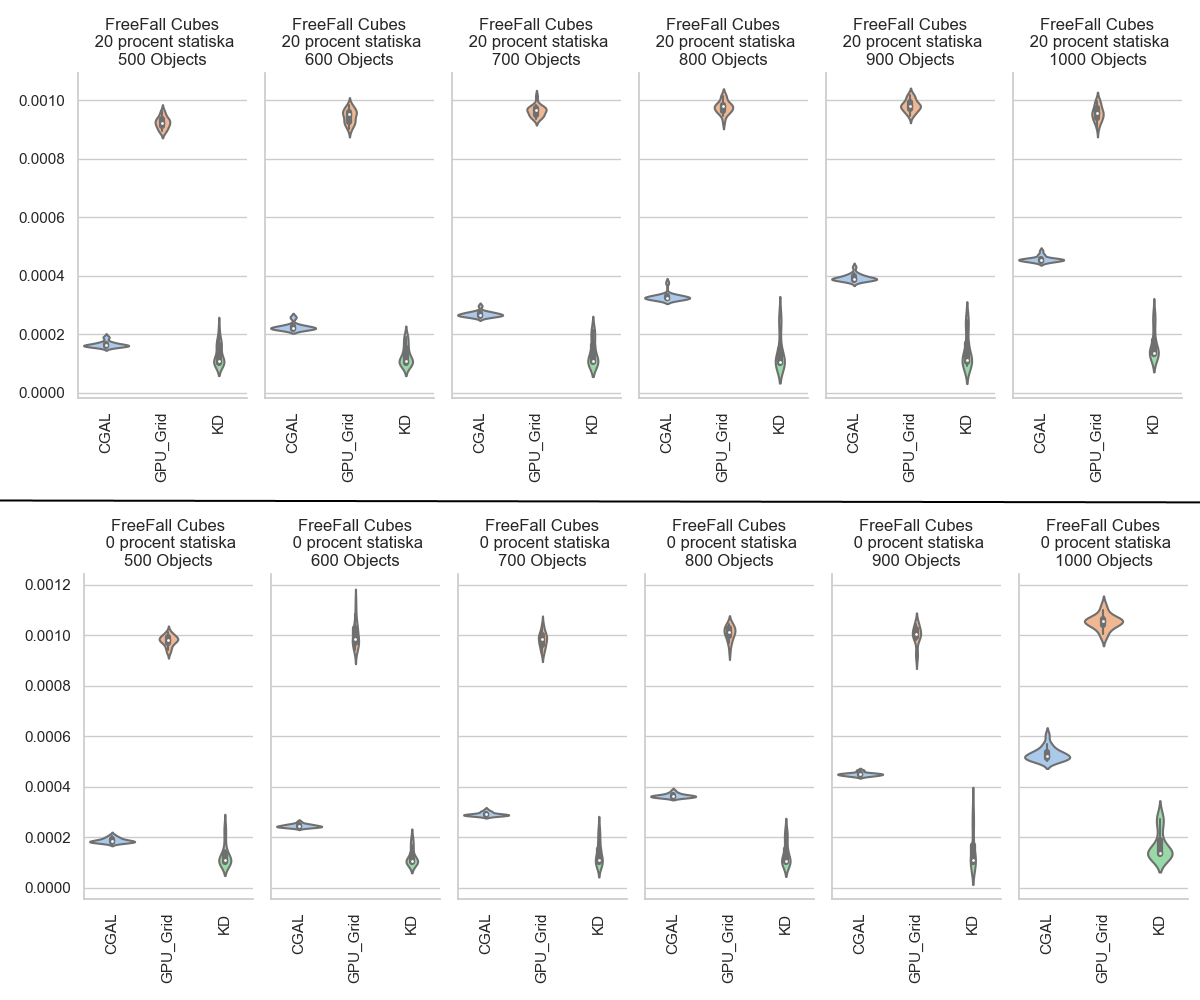
1. Exempel på låddiagram från EJ\_plot, resultatet visar medelvärdets spridning för tre algoritmer prestanda för olika mängder objekt i testscenariot slumpvandring med två olika förhållanden mellan statiska och dynamiska objekt.



1. Exempel på låddiagram från EJ\_plot, resultatet visar medelvärdets spridning för tre algoritmer prestanda för olika mängder objekt i testscenariot fritt fall med två olika förhållanden mellan statiska och dynamiska objekt.



1. Exempel på violindiagram från EJ\_plot, resultatet visar medelvärdets spridning för tre algoritmer prestanda för olika mängder objekt i testscenariot slumpvandring med två olika förhållanden mellan statiska och dynamiska objekt. Samt visar det den statiska sannolikheten för de olika värdena.



1. Exempel på violindiagram från EJ\_plot, resultatet visar medelvärdets spridning för tre algoritmer prestanda för olika mängder objekt i testscenariot frit fall med två olika förhållanden mellan statiska och dynamiska objekt. Samt visar det den statiska sannolikheten för de olika värdena.