SBUF

ID: 12513

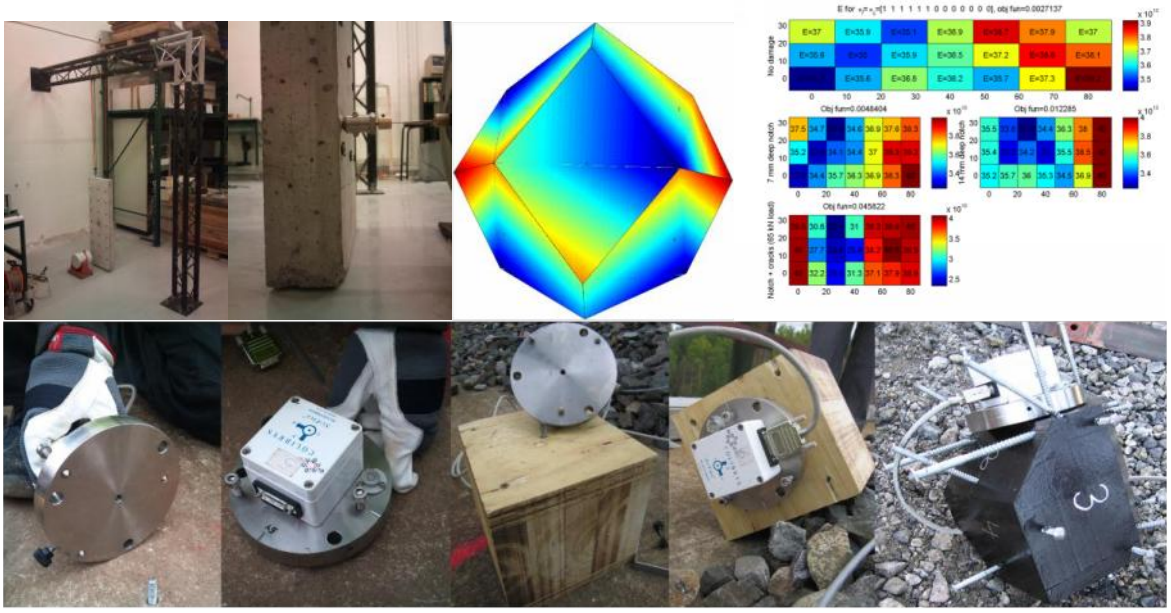
SKANSKA

LULEÅ TEKNISKA UNIVERSITET

FORSKNING OCH UTBILDNING I VÄRLDSKLASS

INTE BARA BROAR

Vibrationsanalys för tillståndsbedömning



Niklas Grip

2013-12-17

SBUF stödjer
forskning & utveckling
som leder till
praktisk handling

FÖRORD

Den här rapporten beskriver en generell analysteknik för tillståndsbedömning av konstruktioner genom vibrationsmätning. Analystekniken har utvecklats så att den är fullt implementerbar i Matlab. Denna rapport innehåller även fullständiga arbetsprocesser och Matlab-skript för att kunna återskapa resultaten inom detta projekt. Analystekniken är i denna rapport verifierad på en generisk betongplatta, ett höghus samt ett antal järnvägsbroar.

Rapportens huvudförfattare är Niklas Grip, men de beskrivna projekten hade ej varit möjliga utan ovärderlig hjälp från Natalia Sabourova, Yongming Tu, professor Lennart Elfgren, Ulf Ohlsson, Thomas Blanksvärd, Fredrik Ljungren och Tarek Edrees vid LTU, samt Andreas Andersson med kollegor från KTH. All mätteknik och experimentella försök hade inte varit möjliga att genomföra utan kollegorna på Complab vid LTU. Vidare vill även huvudförfattaren tacka alla medförfattare till de publikationer som redovisas och refereras till i denna rapport.

Stort tack även till hjälpsamma tips och förslag från Mehdi Bahrekazemi (NCC), Anders Carolin (Trafikverket), Hans Hedlund (Skanska), Raid Karoumi (KTH), Tobias Larsson (NCC), Nils Rydén (PEAB) och Per-Ola Svahn (Skanska) i projektets referensgrupp.

Ett extra tack också till Hans Hedlund och professor Lennart Elfgren som bidrog med tips och råd inför den ansökan som gjorde detta projekt möjligt, samt till SBUF för den ekonomiska stöttningen.

Luleå 17 december 2013
Niklas Grip

Inte bara broar

Vibrationsanalys för tillståndsbedömning

Slutrapport till SBUF-projekt nr 12513

Niklas Grip

Sammanfattning

Behovet av tillståndsbedömning av byggnader och andra konstruktioner har ökat under senare år. Metoder för tillståndsbedömning kan användas för att avgöra om en konstruktion kan fortsätta att bära den last den dimensionerats för eller om den rentav kan bära en ökad last. Alternativt kan den klargöra vilka förstärkningar som behövs för att förlänga byggnadens livslängd till nytta för både miljö och samhällsekonomi.

En resurssnål metod för tillståndsbedömning av en konstruktion bygger på numerisk analys av så kallade fria vibrationer orsakade av naturligt förekommande excitering, som till exempel vågor, vind, människor och trafik. Ett mer resurskrävande alternativ bygger på analys av konstruktionens vibrationssvar på en väl känd konstgjord excitering. Den exciterande kraften måste då mätas, och konstruktionen måste vara isolerad från andra större exciteringskrafter.

I båda fallen finns mät- och analystekniker för att dela upp konstruktionens uppmätta vibrationer i en summa av så kallade vibrationsmoder. Sedan kan numeriska optimiseringsmetoder användas för att justera utvalda nyckelparametrar i en finit elementmodell av konstruktionen så att skillnaden minimeras mellan uppmätta svängningsmoder och de som förutspås av finita elementmodellen. Nyckelparametrarna kan till exempel vara randvillkor och/eller elasticitetsmodul i olika delar av konstruktionen. Lokala variationer av dessa kan då indikera en lokal svaghet i konstruktionen.

Med huvudfinansiering från SBUF har nya vibrationsmätningar utförts på en betongplatta i laboratoriemiljö och på två broar i fält. I båda fallen har vi nyttjat både fria vibrationer och konstgjord excitering. Vibrationsmoder har räknats ut med några olika tekniker och detaljerade finita elementmodeller har gjorts för de olika konstruktionerna. Vi analyserar även vibrationsmoder och jämför mot en finita elementmodell för ett flervåningshus, samt beskriver några faktorer som är speciella för mätning och analys på sådana byggnader.

För betongplattan och en av broarna har mätningarna upprepats efter att ha tillfört olika grader av skada. Analystekniker för tillståndsbedömning har implementerats i MATLAB. I en första utvärdering på betongplattan så är grunda sprickor svåra att detektera medan ökande djup på sprickorna gör att de kan detekteras på rätt halva av plattan eller mer noggrant. MATLAB-koden struktureras för närvarande om för att publicera i en form lämplig för andra att anpassa till egna liknande tillämpningar. MATLAB-koden skall i nästa skede anpassas för att utvärdera tillståndsbedömningen på de omfattande mätningar som genomförts i samband med fullskaleförsök på en 33 m lång stålfackverksbro. Tillståndsbedömningen innefattar mätningar från både intakt bro, bro med delar skadade samt efter att bron belastats till brott.

Dessutom har enkel utrustning och enkla analysmetoder utvecklats för att säkerställa snabb och säker kalibrering av accelerometrar i fält. Detta behövs för att få jämförbara mätsignaler från de olika accelerometrarna oavsett ändringar i temperatur och väderförhållanden.

Innehåll

1 Bakgrund

Behovet av tillståndsbedömning av byggnader har ökat stadigt i Europa, USA och Asien de senaste årtiondena. Såväl bristfällig konstruktion som tärande omgivning kan snabba på vittring, sprickor och andra försvagningar av till exempel pelare, väggar, plattor, bjälkar och grund i åldrande byggnader. Bra metoder för tillståndsbedömning kan då användas för att godkänna en konstruktion för fortsatt eller ökad belastning, alternativt avslöja vilka förstärkningar som behövs för att förlänga byggnadens livslängd. Detta främjar både miljö och samhällsekonomi.

Mätningar och analys för tillståndsbedömning är traditionellt både dyra och tidskrävande. Några anledningar till detta är dyr mätutrustning, mätningar som kräver mycket planering och personal, samt analystekniker som finns beskrivna i forskningsartiklar, men är mycket tidskrävande att implementera och ej heller tillgängliga som fri programvara.

Allt eftersom prisnivån på robusta sensorer och annan teknologi går ned så ökar dock möjligheterna att bygga in sensorer för kontinuerlig mätning och analys. På sikt vore det därför önskvärt med metoder och lättillgänglig programvara som automatiskt identifierar viktiga nyckelvärden och helst redan på ett tidigt stadium varnar när en konstruktions självsvängningar ändras på ett sätt som kan indikera behov av någon form av underhåll.

1.1 Svängningsmoder med tillhörande egenfrekvenser och dämping

Alla byggnadskonstruktioner har olika egenfrekvenser, var och en förknippad med en typ av egensvängning, eller svängningsmod. Om konstruktionen utsätts för krafter vars variationer sammanfaller med en egenfrekvens så uppstår resonans och konstruktionen sätts i egensvängning. Det kanske mest välkända klassiska exemplet är Tacoma Narrows Bridge i Washington, som hade en dåligt dämpad vridningsmod med periodlängd fem sekunder, eller egenfrekvens 0,2 Hz. Bara fyra månader efter brons invigning aktiverades den egensvängningen av vinden och växte sig efterhand så stark att bron kollapsade november 1940, se Figur 1.

Modernare konstruktioner är självklart konstruerade så att alla egensvängningar är rejält dämpande, men varje konstruktion kännetecknas av olika egensvängningar med tillhörande egenfrekvens och dämpning som kan räknas ut, till exempel från en finita elementmodell av konstruktionen. Om man utsätter konstruktionen för krafter med variationer som inkluderar olika egenfrekvenser så aktiveras motsvarande vibrationsmoder. Med accelerometrar utplacerade på konstruktionen kan lokala vibrationer mätas och det finns olika analystekniker för att dela upp de uppmätta självsvängningarna i svängningsmoder och räkna ut dessas vibrationsfrekvens och dämpning.

Detta illustreras i Figur 2 (a), där en förenklad endimensionell ögonblicksbild visar hur en brobanas uppmätta eller predikterade egensvängningar i en given riktning (överst) kan delas

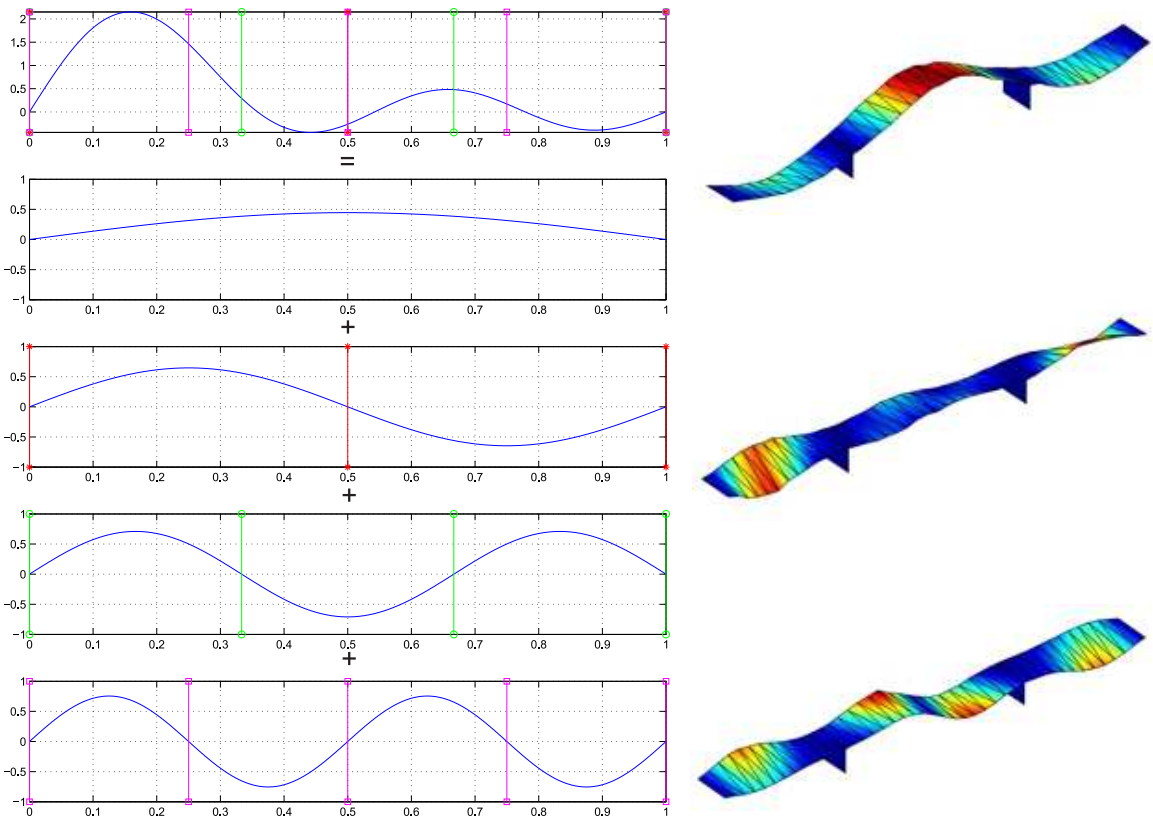
Figur 1: Tacoma Narrows Bridge. Se även http://www.youtube.com/watch?v=3mclp9QmCGs.

upp i en summa av enklare svängninsmoder (plot 2–5) med olika modform, amplitud och svängningsfrekvens. Vill man kunna detektera dessa fyra svängningsmoder så är en bra tumregel att accelerometrarna inte placeras i någon punkt där en svängningsmod alltid har amplitud 0, här markerat med lodräta streck. Ett enkelt sätt att finna bra mätpunkter är att multiplicera alla modformer av intresse och välja mätpunkter där den produktern har stora värden.

1.2 Uppdatering av finit elementmodell (FEM-uppdatering)

Uppmätta svängningsmoder kan utnyttjas för att justera utvalda nyckelparametrar i en finita elementmodell på så vis att de självsvängningar som förutsägs av modellen stämmer bättre överens med uppmätta svängningsmoder samt deras egenfrekvenser och dämpning. Manuell sådan justering görs ofta för att justera värden på dåligt kända materialparametrar eller randvillkor så att modellen stämmer bättre med verkligheten.

Detta går att ta några steg längre genom att konstruera datorprogram som använder någon metod för numerisk optimering för att automatiskt justera utvalda nyckelparametrar för så bra överensstämmelse som möjligt mellan föruttspådda och uppmätta egensvängningar. Om de utvalda parametrarna till exempel är materialets elasticitetsmodul i olika delar av konstruktionen så kan en lokal svaghet, till exempel en spricka, detekteras genom att modellen, som ej har någon spricka där, justerar ned elasticitetsmodulen till ett lägre värde än det verkliga för att förutspådda självsvängningar ändå skall stämma så bra som möjligt överens med de uppmätta. Lokala defekter kan alltså på detta sätt upptäckas genom lokal sänkning av elasticitetsmodul,



(a) (b)

Figur 2: (a) Uppdelning av en brobanas svängningar i en summa av svängningsmoder. Produkten av dessa kan tydliggöra lämpliga mätpunkter för modalanalys. (b) Modformer för verklig bro.

alternativt sänkning jämfört med tidigare mätningar på samma konstruktion.

1.3 Fria vibrationer eller konstgjord excitering

Om man vill göra tillståndsbedömning baserad på uppmätta egensvängningar så är ett viktigt första val om man skall utgå från naturligt förekommande vibrationer som orsakas av omgivningen eller om man skall sätta konstruktionen i svängning med en känd insignal och bestämma egensvängningar utifrån uppmätt insignal-utsignalförhållande.

I det senare fallet så tillkommer extra tidsåtgång och kostnader, dels för att på ett bra sätt excitera självsvängningar i konstruktionen, till exempel med en så kallad "shaker" (Figur 4 sid 10, Figur 23 sid 32) eller impulshammare. Detta kan vara opraktiskt för byggnader där folk bor eller vistas. Dessutom kan "fel" typ av excitering ge svårtolkade mätdata. Till exempel kan ett tungt lastat tåg belasta en bro så att en löst inspänd ände av brobanan delvis lättar från fundamentet[a], vilket markant kan ändra konstruktionens randvilkor och dynamiska respons. Det blir då väldigt svårt eller omöjligt att dra några slutsatser från jämförelse av uppmätta svängningsmoder mot till exempel tidigare uppmätta eller eller svängningsmoder förutspådda av en finita elementmodell av bron. Desutom måste den exciterade kraften mätas och finns det andra icke försumbara vibrationskällor som kan störa mätningarna så måste konstruktionen isoleras från dessa, så att man har en enda väl känd insignal och mäter vibrationssvaret på denna med utplacerade accelerometrar.

Det finns gott om stora konstruktioner som naturligt utsätts för många olika slumpartade vibrationer och krafter från omgivningen, till exempel från vågor, vind, trafik och/eller människor. Sådana vibrationer orsakade av den omgivande miljön kallas för *fria vibrationer* (på Engelska *ambient vibrations*). Då kan det vara betydligt enklare att använda så kallad *operationell modalanalys* (på Engelska *operational modal analys*) [BVA03], som mäter vibrationssvaret på en *okänd* insignal och räknar ut modformer, frekvenser och dämpning från dessa. Man slipper då lägga tid och resurser på exciteringen, men samtidigt krävs att den naturliga exciteringen från omgivningen är sådan att den exciterar svängningsmoder av intresse utan att tillföra andra oönskade effekter. Att enbart excitera en frekvens eller punkt i konstruktionen är till exempel ej tillräckligt för operationell modalanalys. Istället vill man ha excitering i många punkter och många frekvenser. Detta förklaras närmare till exempel i [Bri13]. Några exempel på bra exciteringskällor som anges där är vindbelastning på stora konstruktioner, vibrationer från trafik eller mäniskor, slumpartade belastningar som rör sig över konstruktionen samt stora maskiner med många rörliga delar. Samtidigt gäller att inte ha så stora exciteringskällor att man mäter exciteringskällan snarare än fria vibrationer [Ric97], eller så stora laster att de ger avvikelser från det linjära system som en finita elementmodell räknar på, till exempel ickelinjära effekter när delar av konstruktionen slamrar och slår mot varandra.

1.4 Huvudsteg för tillståndsbedömning via FEM-uppdatering

Hela kedjan från mättillfälle till utsaga om eventuella defekter kan delas in i följande huvudsteg.

1. Gör en finita elementmodell av konstruktionen, och välj ut nyckelparameter som kan varieras, som till exempel elasticitetsmodul och/eller randvillkor i olika delar av konstruktionen.

2. Placera ut accelerometrar i lämpliga mätpunkter.

[a]För ett belysande exempel, se hur mycket passerande tåg kan få Manhattan Bridge kan svaja, se http://www.youtube.com/watch?v=DgXveBf_l6k.

(Dessa kan till exempel väljas genoma att använda tumregeln i Figur 2 (a) för de modformer som förutsägs av finita elementmodellen.)

3. Mät konstruktionens självsvängningar. (Fria vibrationer eller konstgjord excitering.)

4. Modalanalys för att skriva om uppmätta svängningar som en summa av olika svängningsmoder med tillhörande svängningsfrekvens (eventuellt även dämpning).

5. Utveckla och använd programvara för kalibrering av de utvalda nyckelparametrarna så att förutspådda och uppmätta svängningsmodformer, egenfrekvenser och eventuellt även dämpningar stämmer så bra som möjligt överens med uppmätta dito.

Den här rapporten beskriver forskning med huvudfinansiering från SBUF och Formas för att undersöka, utveckla och utvärdera den mätteknik och analysmetoder som ingår i de olika steken ovan. Varje steg i kedjan är viktig, men vi kommer framför allt att beskriva delar av steg 3, 4 och 5. För hela kedjan till tillståndsbedömningen redovisar vi några preliminära resultat för en betongplatta i laboratoriemiljö. Så långt är ingen del av framställningen knuten till någon speciell typ av konstruktion eller byggnad. Vi beskriver även pågående arbete för att använda samma metoder i större skala på en järnvägsbro, men även där är tillvägagångssättet samma för andra byggnadskonstruktioner.

Ett slutmål med projektet, som nu fortsätter med finansiering från Formas, är att göra mät- och analysmetoder enklare och mer lättillgängliga, bland annat genom att göra utvecklad programkod fritt tillgänglig.

1.5 Syfte

Då tidigare fältmätningar för modalanalys gjorts på olika broar i samarbete med LKAB och Trafikverket så var ett syfte med den kompletterande finansieringen att möjliggöra fortsättning av dessa forskningsprojekten, men med mer fokus på delprojekt som inte är specifikt knutna till just broar.

Bland annat innefattar detta att som ett första steg utföra mätningar, modalanalys och tillståndsbedömning i labmiljö på en betongplatta med olika grader av tillförd skada. Det ingår även att vidareutveckla metoder för snabb och säker accelerometerkalibrering, samt att komplettera analys på broar med alanalys på någon större byggnad av annan typ och undersöka vilka faktorer som är speciellt viktiga för mätning och analys på sådana byggnader.

1.6 Mål

Ett inledande mål var att implementera och testköra MATLAB-kod för tillståndsbedömning på betongplatta med olika grader av skada. Detta som första steg för att sedan generalisera koden för större konstruktioner. Ett viktigt slutmål för projektet som helhet är att efter hand göra utvecklad programkod för de olika metoderna tillgänglig för nedladdning. Som ett första steg i den riktningen skulle delar av koden dokumenteras och publiceras i samband med slutrapporten.

För att ytterligare minska kopplingen just till broar ville vi även utföra modalanalys på någon annan större byggnad och dra slutsatser om viktiga skillnader vid modalanalys på olika typer av stora byggnader.

Hela kedjan från mättillfälle till tillståndsanalys är viktig, så fokus har också varit på att förbättra mätningarna, speciellt att vidareutveckla mät- och och analysmetoder för enkel och pålitlig kalibrering av accelerometrar så att mätningar från olika accelerometrar och mättillfällen blir direkt jämförbara.

1.7 Metod

Nya vibrationsmätningar har utförts på betongplatta i labmiljö samt på två olika broar. För betongplattan och en av broarna har mätningarna upprepats efter tillförsel av först olika små skador och sedan belastning till brott. För dessa mätningar samt mätningar på en byggnad vid Stockholms central har sex olika metoder för modalanalys använts och jämförts.

För Betongplattan har tillståndsanalys med en metod för FEM-uppdatering implementerats i MATLAB. Koden håller för närvarande på att struktureras om för att bli enklare och mer lättillgänglig, och skall även generaliseras och utvärderas för de mätningar som gjorts på samma bro med olika nivåer av skada.

För mätningarna i fält har en enkel mätmetod, mäthjälpmedel och analysmetod utvecklas som möjliggör pålitlig accelerometerkalibrering enbart från snabbt genomförda mätningar av jordens gravitationskraft.

De olika delprojekten beskrivs översiktligt i avsnitt 2. Diskussion och slutsatser följer i Avsnitt 3, följt av pågående och fortsatt forskning i Avsnitt 4. Avslutningsvis följer några publicerade artiklar och konferensbidrag i separata appendix.

2 Projektbeskrivning

Vi beskriver ingående delprojekt under separata rubriker nedan. De analysmetoder som valts för modalanalys och tillståndsbedömning beskrivs i Avsnitt 2.1. Utvärdering av dessa på en betongplatta i laboratoriemiljö beskrivs i Avsnitt 2.2. Avsnitt 2.3 beskriver mätning och modalanalys dels på ett flervåningshus, och dels på en järnvägsbro före och efter tillförsel av först mindre skador och sedan belastning till brott. Förbättrade och förenklade metoder för accelerometerkalibrering vid mätningar i fält förklaras i Avsnitt 2.4.

2.1 Modalanalys och FEM-uppdatering

För modalanalys av fria vibrationer har vi än så länge använt det kommersiella programmet ARTeMIS[b] Vi har framför allt varit speciellt intresserade av följande analysmetoder:

1. Stochastic Subspace Identification (SSI) [OM96, PDR99, PDR01, RDR08, RPDR08], som analyserar i "tidsdomänen".

2. Frequency domain decomposition (FDD, se t ex [BZA01]), som analyserar frekvensinnehållet i mätsignalen med hjälp av singulärvärdesuppdelning för att kunna hålla isär även svängningsmoder som ligger nära varandra i frekvens.

3. Wavelet-baserade metoder för att analysera hur frekvensinnehållet varierar med tiden (som illustrerat med ljud och bild i [EGOS09]). Dessa sägs också ha som fördel att kunna hålla isär även svängningsmoder som ligger nära varandra i frekvens (se t ex [CLL09]).

SSI och några varianter av FDD är de som ingår i ARTeMIS och har använts för de projekt som beskrivs i den här rapporten samt även i en inledande studie på på en bro över järnvägen i Kiruna [Sab11]. Dessutom planeras för närmare jämförelse mellan tid- och frekvensupplösningen för de två senare, se Avsnitt 4.4.

För tillståndsbedömning har vi implementerat en metod för FEM-uppdatering som beskrivs i [TMDR02, TDR03, RTDR10]. Den använder sig av en speciell så kallad *målfunktion* (*objective*

[b] http://www.svibs.com/

function), som ger ett mått hur stor skillnaden är mellan uppmätta och förutspådda modformer och frekvenser. Värdet på denna minskas genom att stegvis ändra utvalda nyckelparametrar med en numerisk optimiseringsmetod (*Newton trust region* [NW06] implementerad med hjälp av kommandot `fmincond` i MATLABs Optimization Toolbox). För att ändra parametervärden i rätt rikting används uppskattningar från [FK68]. När parametervärdena ändrats anropas FEM-rogrammet Abaqus för att räkna ut nya förutspådda svängningsmoder. Detta ger det flödesschema som visas i Figur 3. För den koden fås analysresultaten i nästa avsnitt relativt snabbt med en vanlig bordsdator. För större strukturer håller vi på att generalisera koden och förbereda en datorkluster på 96 kärnor för den parallellkörning av MATLAB och Abaqus som krävs (se Avsnitt 4.1).

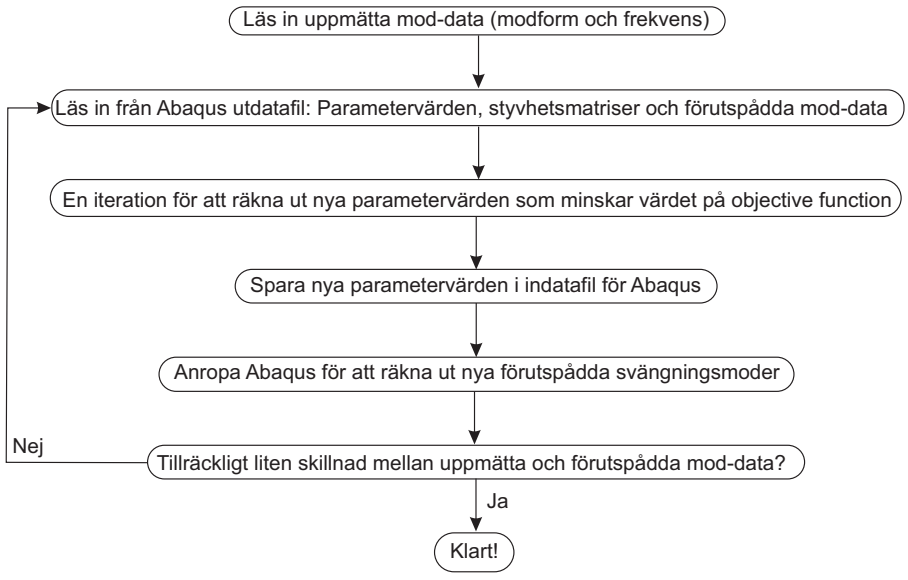
2.2 Laboratoriemätningar på betongplatta

Som ett första steg för att utvärdera metoder för FEM-uppdatering gjorde vi i laboratoriemiljö mätningar på en fritt upphängd armerad betongplatta som visas i Figur 4. Plattan är av storlek $1\,050 \times 340 \times 70$ mm med tre längsgående armeringsjärn placerade enligt Figur 5.

Som en första jämförelse så utfördes modalanalys både på fria vibrationer och på frekvenssvaret som ficks från mätningar med excitering från skakern i Figur 4. För fria vibrationer exciterades plattan med slumpartat försiktigt knackande fingertoppar på plattans baksida. Detta gav inga egentliga skillnader i resulterande modformer och frekvenser. För det fortsatta arbete som beskrivs nedan exciterades plattan enbart med shakern. Plattan var fritt upphängd (se Figur 4) för att isolera den från andra vibrationer och säkerställa att de exciterade krafterna från shakern var de enda av betydelse. Från mätningar av den exciterande kraften och enaxliga accelerometrar placerade i $5 \cdot 13 = 65$ punkter kunde plattans frekvenssvar i varje sådan punkt räknas ut. Sedan kunde modformer och frekvenser räknas ut med standardmetoder som speciellt studerar frekvenssvarets imaginärdel, se till exempel [Ric97] och [CJK06, Avsnitt 18.6.1].

För att kunna utvärdera metoder för tillståndsbedömning så upprepades sedan samma mätningar med shaker efter att ha tillfört olika grader av skada vid den armerade sidan av plattan med totalt fem olika skadenivåer:

1. Oskadad platta.

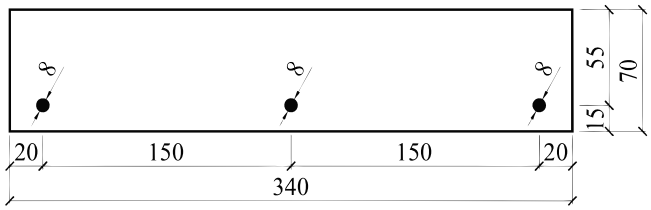


Figur 3: Flödesschema för implementerad metod för FEM-uppdatering.

2. Sågad 7 mm djup skåra.
3. Samma skåra sågad till djup 13.5 mm.
4. Djupare spricka kring skåran skapad med linjeformad last på 6,6 kN (se Figur 6 (a)).
5. Ännu djupare sprickor (Figur 6 (b)) skapade genom att manuellt med tvingar lägga på ytterligare linjär last.

Mätningarna gjordes i samarbete med Fredrik Ljungren på Avdelningen för akustik vid Luleå tekniska Universitet, som hade lämpliga erfarenheter, shaker och mätutrustning för just dessa mätningar. Figur 7 visar uppmätta modformer för plattan med 7 mm djup skåra.

Figur 4: Mätningar på fritt hängande armerad betongplatta.

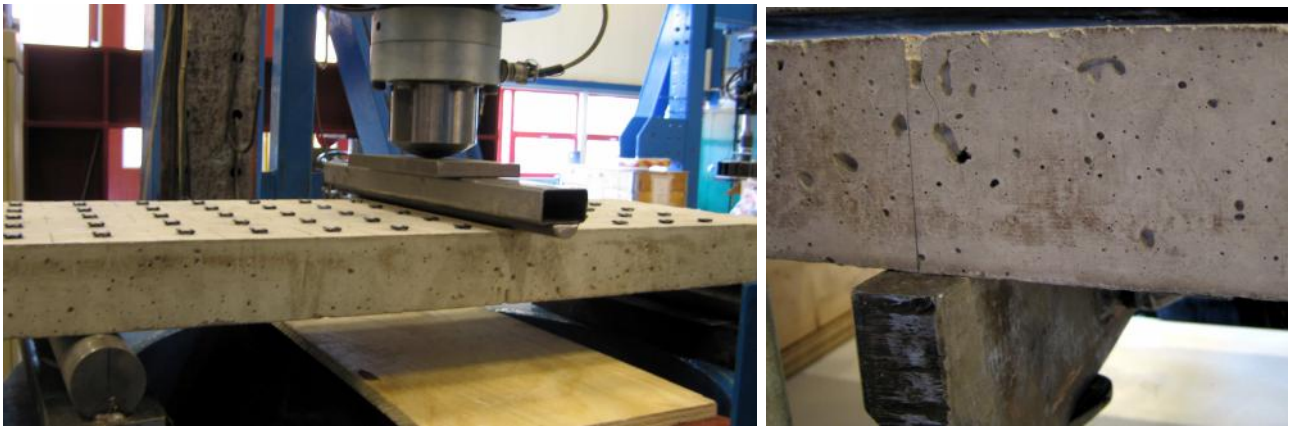


Figur 5: Tvärsnitt av plattan (enhet: mm).

Gästprofessor Yongming Tu från Kina har gjort finita elementmodeller av plattan, där vi som mest har delat in plattan i $13 \times 5 = 65$ grupper av finita element med en separat elasticitetsmodul för varje grupp, se Figur 8.

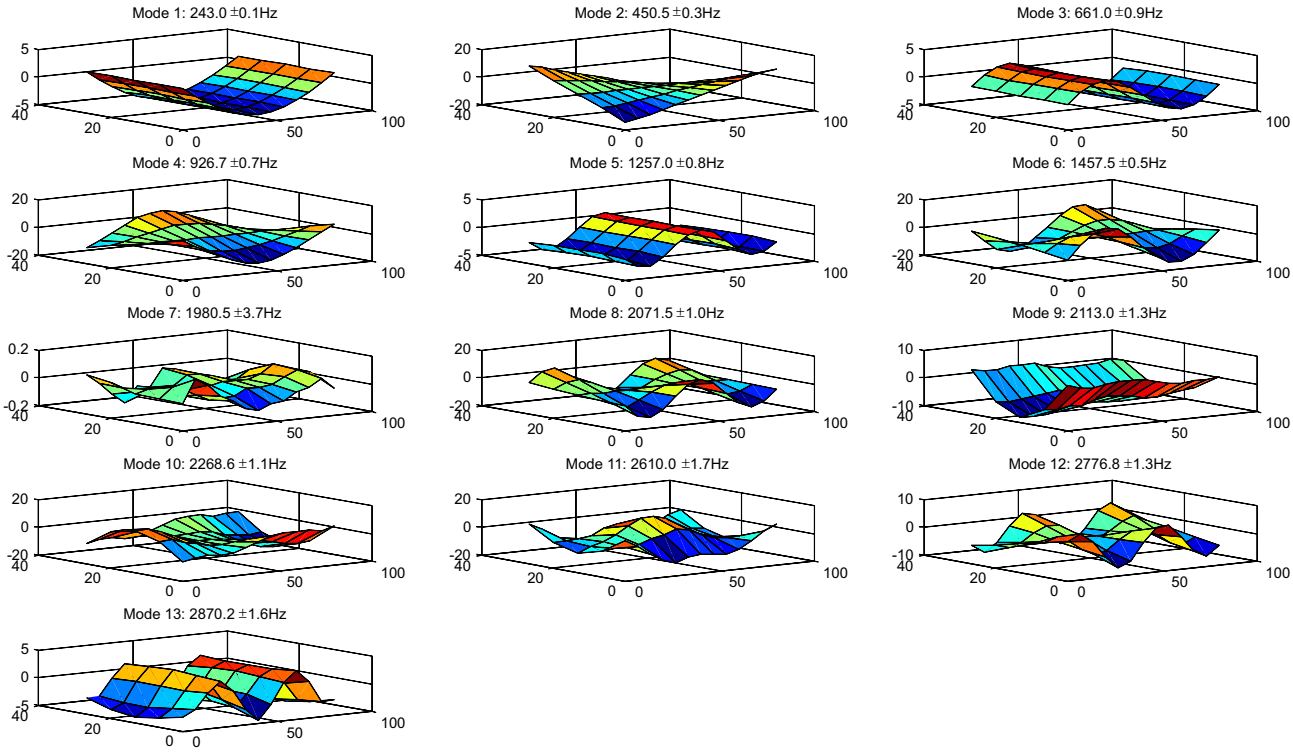
Nästa steg var att testa tillståndsbedömning för betongplattan med vår MATLAB-kod för FEM-uppdatering. Denna justerar elasticitestmodulen [GPa] i en modellerad oskadad platta så att dess modformer och -frekvenser i så hög grad som möjligt sammanfaller med de uppmätta. Oavsett vilka slumpvis valda startvärden vi hade på de 65 elasticitetsmodulerna så konvergerade koden till de resultat som visas i Figur 9.

Bredden på skåran bör inte påverka resultaten märkbart, utan vi förväntade oss att skårans eller sprickans djup är den faktor som främst påverkar plattans modformer och frekvenser. Man



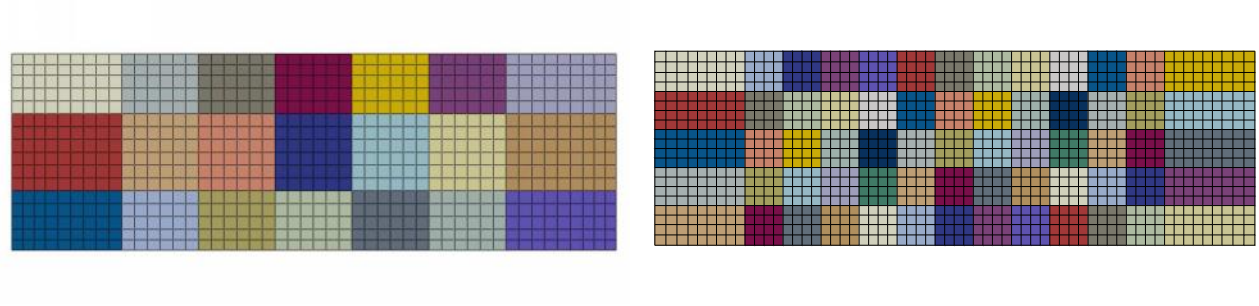
(a) (b)

Figur 6: (a) Kontrollerad böjbelastning vid uppsågad skåra. (b) Synlig djupare spricka efter ytterligare böjbelastning.



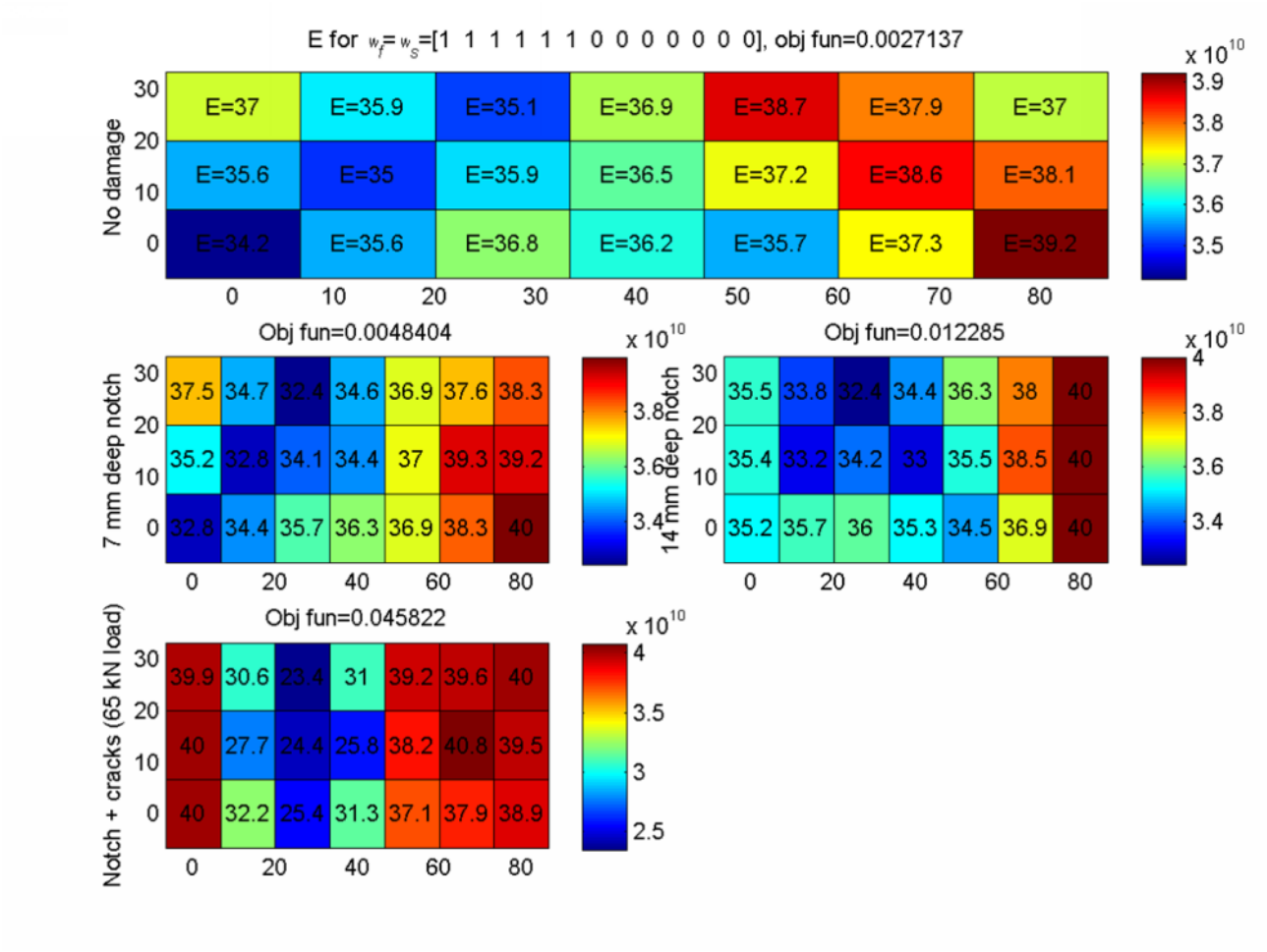
Figur 7: Modformer för plattan i fallet med 7 mm djup sågad skåra.

kan alltså förvänta sig att djupare skåra/spricka skall markeras tydligare i Figur 9. Man kan där se en aning lägre elasticitetsmoduler i vänstra halva redan för den oskadade plattan (överst). Detta skulle till exempel kunna bero på att den halvan är 1–2 mm tunnare för den verkliga



(a) (b)

Figur 8: Grupper av element för FE-modell för betongplattan. (a) 16×48 element indelade i 3×7 grupper. (b) 20×62 element indelade i 5×13 grupper.



Figur 9: FEM-uppdatering för betongplattan justerar elasticitestmodulen [GPa] i en modellerad oskadad platta så att dess modformer och -frekvenser i så hög grad som möjligt sammanfaller med de uppmätta.
Jämfört med oskadad platta (överst) så ger sprickor med tilltagande djup en gradvis övergång till lägre värden kring sprickan.

plattan.

I mittenraden i Figur 9 är det svårt att med ögat se någon skillnad i slutresultatet för 7 mm djup skåra (vänster), men för 13,5 mm djup skåra kan man se en lite mer tydlig övergång till lägre elastititetcmodul på vänstra halvan.

Nederst i Figur 9 visas analysresultatet efter knäckning till djupare spricka med 65 kN. Nu syns en tydlig koncentration av låga värden på den delen av plattan där sprickan är belägen.

Analysresultaten tyder alltså på att en 3,5 mm djup spricka inte märkbart ändrar den armerade plattans egensvängningar. För 13,5 mm djup skåra kan man se indikation av försvagning i vänster halva, och för den djupare sprickan indikeras mer tydligt och exakt var på plattan skadan är belägen.

2.3 Mätningar och analys på större konstruktioner

Vi redogör här för fältmätningar och analys som utförts på en större byggnad i Stockholm samt en järnvägsbro över Åby älv.

2.3.1 Modalanalys på flervåningsbyggnad

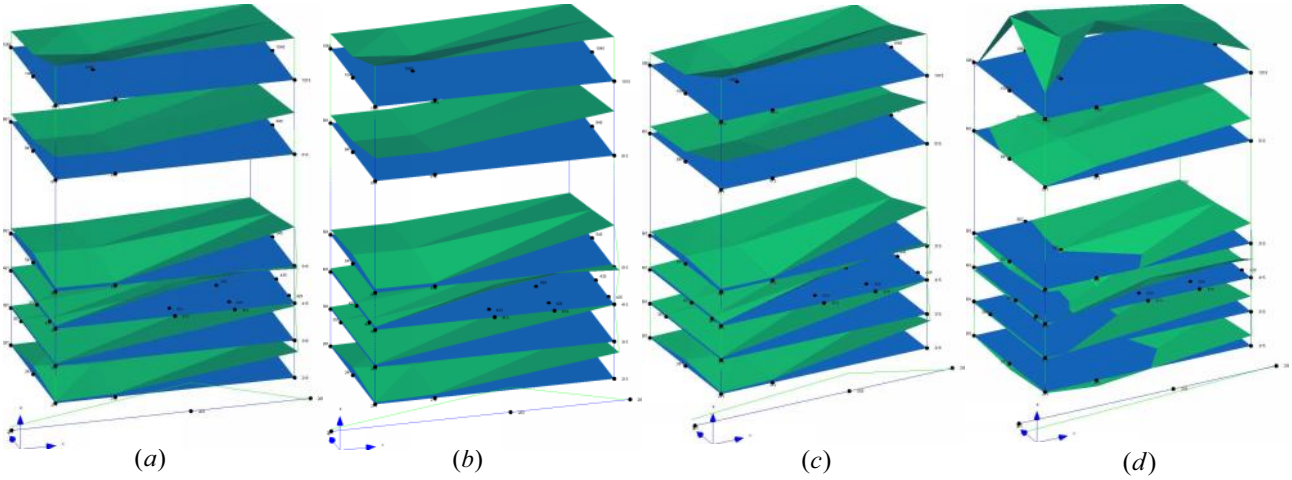
Kvarteret Klassföreståndaren färdigställdes 2008 mellan Stockholms centrals norra bangårdsområde och södra delen av Vasastan i centrala Stockholm. Hyresgäster på plan 10 klagade dock på vibrationer som förmodades vara orsakade av förbipasserande tåg. Olika typer av accelerometermätningar utfördes under ledning av P-O Svahn, Skanska Sverige AB [Sva10] för att undersöka hur stora dessa vibrationer var. En *första serie mätningar* gjordes med excitering från passerande tåg. En *andra mätserie* utfördes med en mätvagn på rälsen som kan excitera en sinusformad last med valfri frekvens medan vagnen är stilla eller i rörelse. För stillastående vagn kan byggnadens svängningsmoder beräknas från analys av byggnadens frekvenssvar (för mätvagnen) eller med metoder för operationell modalanalys för vibrationer orsakade av förbipasserande tåg. Vi har valt att analysera de vibrationer som orsakas av förbipasserande tåg. Detta ger bara mätdata från 15 mätpunkter, men efter ompositionering av 14 accelerometrar till nya positioner i byggnaden så utfördes en *tredje mätning*. Därmed kan accelerometern som var på samma ställe vid båda mätningarna användas som referensaccelerometer. och vi har ett större antal mätpunkter utspridda på sex plan i byggnaden. Det är fortfarande få accelerometrar per plan, så det duger inte för en noggrann bestämning av modformer, men det räcker för att bestämma modsvängningarnas frekvens.

Ett problem med den tredje mätomgången var dels att mätningarna var relativt korta, och dels gjordes de med konstgjord excitering inne i byggnaden. Då finns stor risk att man mäter de exciterande vibrationerna lika mycket som byggnadens fria vibrationer. Att enbart analysera avklingande vibrationer efter att exciteraren slagits av var heller inget alternativ då detta var en kort signal, med liten amplitud och sparad med enbart 16 bitars upplösning vilket gav stora avrundningsfel. Varken detta eller att använda hela signalen gav därför några användbara resultat. Istället utnyttjade vi att exciteraren gav en svepande sinussignal med frekvens ökande från 4 till 16 Hz, genom att analysera den senare halvan av signalen och lägga på ett digitalt lågpassfilter (raised cosine av grad 4) som behåller alla frekvenser upp till 9 Hz[c] och filtrerar

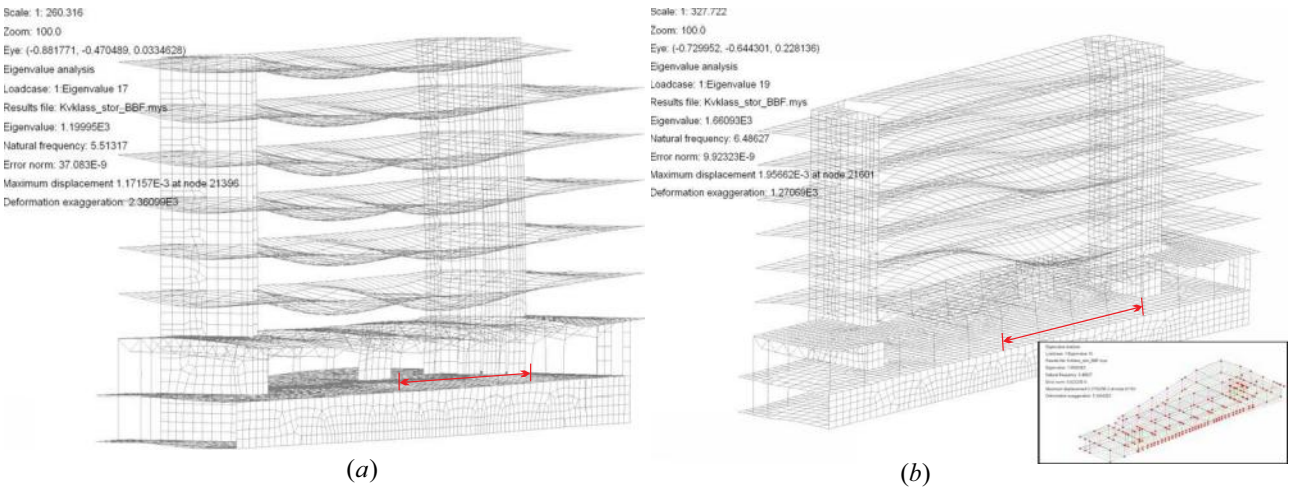
[c]Ett alternativ för eventuella senare jämförelser kan vara att använda hela signalen och räkna ut modform från denna och från exciterande krafter genom analys av byggnadens frekvenssvar [CJK06, Avsnitt 18.6.1]. För första accelerometersetupen kan i så fall modform räknas ut som vi gjort eller även där via analys av frekvenssvar för excitering med mätvagnen. För metoden som beskrivs i [CJK06, Avsnitt 18.6.1] är det då inte helt enkelt att identifiera vilka toppar i frekvensspektrumet som hör till vibrationsmoder. Vi valde därför att analysera fria vibrationer med ARTeMIS.

bort allt med frekvens över 12 Hz.På så vis fick vi en mätsignal utan störande exciterare, lång nog för modalanalys i ARTeMIS och med nyligt exciterade vibrationer i frekvensbandet 0-9 Hz, där vi har förutspådda vibrationsmoder från en finita elementmodell att jämföra mot.

I ARTeMIS finns sex metoder för modalanalys. De tre första är tre olika varianter av medoden "Frequency Domain Decomposition" (FDD) de fungerar bra vid andra mätningar vi gjort (se till exempel Figur 28, sidan 36), då normalt med 20-60 minuters mättid och därmed högre signal-brusförhållande i spektrumet. Mätningarna på den här byggnaden var bara någon minut långa och då hittade FDD inga vibrationsmoder i det frekvensintervall där vi har förutspådda moder att jämföra mot. De återstående tre metoderna är olika varianter av "Stochastic Subspace Identification" (SSI). De hittade vibrationsmoderna med lägst frekvens för två av SSI-varianterna visas i Figur 10. Frekvenserna stämmer i det första fallet någorlunda



Figur 10: Resultat av modalanalys för mätningar från accelerometrar på plan 2, 3, 4, 5, 8 och 10: För analysmetoden "SSI Unweighted Principial Components" var de lägsta modfrekvenserna (a) 5.83 Hz och (b) 6.82 Hz. För "SSI Weighted Principial Components" var de lägsta modfrekvenserrna (c) 6.00 Hz (d) 7.40 Hz

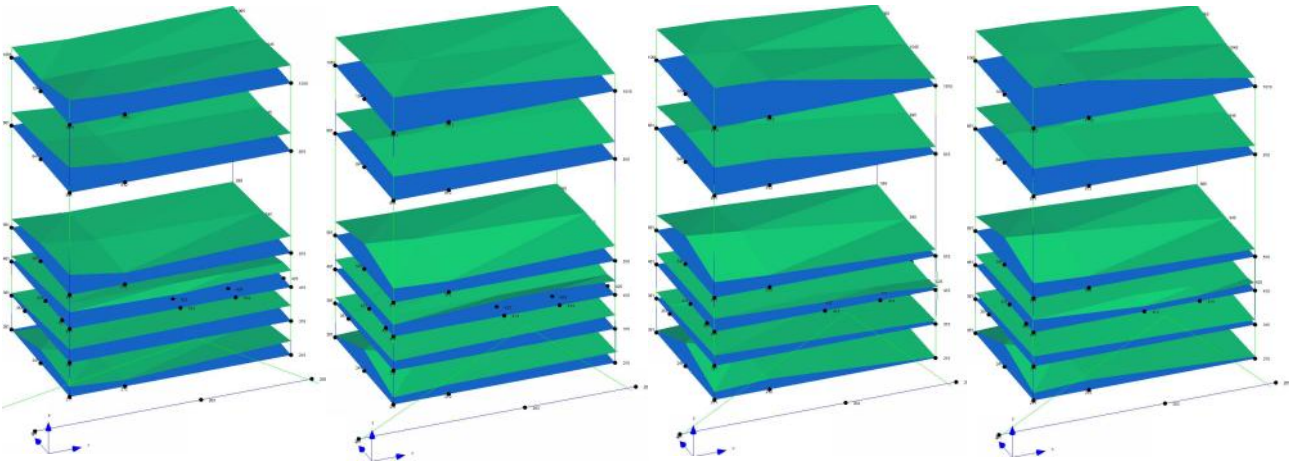


Figur 11: Vibrationsmoder för byggnaden beräknade ur finita elementmodell i [Sva10]. Den röda pilen indikerar vilken del av byggnaden som analyserats i Figur 10 och 12. (a) Första svängningsmod med koppling mellan grund-stomme, $f_0 = 5,5$ Hz. (b) Andra svängningsmod med koppling mellan grund - stomme, $f_0 = 6,5$ Hz.

mot de förutspådda frekvenserna 5,5 och 6,5 Hz i Figur 11, men modformerna avviker från de förutspådda, då modformernas maximum ligger i fel ända av varje våningsplan. För den tredje metoden "SSI Canonical Variate Analysis" visas de fyra första funna modformerna i Figur 12. De två med lägst frekvens stämmer relativt väl mot de förutspådda frekvenserna 5,5 och 6,5 Hz. Buktningen av motsvarande modformer i Figur 11 syns ej lika klart i Figur 12, men det är för få accelerometrar för att tydligt mäta modformen och för punkter som saknar accelerometer har modformens värde där räknats ut med linjär interpolation.

Några viktiga slutsatser för fullskalig modalanalys och tillståndsbedömning för sådana byggnader är att

- Jämfört enklare konstruktioner som broar så behövs vibrationsmätningar i betydligt fler mätpunkter, speciellt för stora byggnader med flera våningsplan.
- Till skillnad från broar så kan accelerometrarna placeras så att de står inlåsta i olika rum. Detta underlättar att mäta fria vibratoner i ett stort antal mätpunkter med lång tid för varje mätpunkt utan att behöva ett stort antal accelerometrar. Det räcker nämligen att ha någon referensaccelerometer placerad på samma ställe vid varje mätning och flytta runt övriga tillgängliga accelerometrar till man har mätningar i alla mätpunkter av intresse.
- Analys med Stochastic Subspace Identification klarade korta mätsignaler med lågt signal-brusförhållande bättre än motsvarande modalanalys med Frequency Domain Decomposition.



Figur 12: Samma analys som i Figur 10 men med metoden "SSI Canonical Variate Analysis". De lägsta funna modfrekvenserna var (a) 5.7 Hz, (b) 6.5 Hz, (c) 7.0 Hz och (d) 7.42 Hz.

2.3.2 Mätningar och analys på järnvägsbro före och efter skada

En inledande jämförelse av samma tekniker för modalanalys som i Avsnitt 2.3.1 gjordes i samarbete med LKAB i samband med mätningar på en bro över järnvägen i Kiruna [Sab11]. Sedan har mätningar gjorts på två olika järnvägsbroar i ett samarbete med Trafikverket. En över Långforsen, och en över Åby älv, se Figur 13. Vi beskriver dessa närmare i Appendix B. För bron över Åby älv upprepades mätningarna fem gånger. En mätomgång medan bron var i bruk, en sedan den lyfts upp på temporära stöd land, två mätserie efter att ha tillfört två

grader av liten lokal konstruerad skada och slutligen en femte mätomgång efter att ha belastat bron till brott.

Exempel på modalanalysresultat och jämförelse mot vibrationsmoder förutspådda av finita elementmodell finns i Appendix B. MATLABkoden vi använde för FEM-uppdatering på betongplattan i Avsnitt 2.2 håller vi på att modifiera för att kunna utvärdera samma analysmetoder på bron med olika grader av skada.

De flesta problemen som behöver hanteras är dock inte speciellt knutna till broar, utan handlar främst om generalisering till större konstruktioner med mer komplicerad geometri (se dock punktlistan sist i Avsnitt 2.3.1).

Figur 13: Mätningar av självsvängningar har gjorts på två järnvägsbroar. En över Långforsen (övre bilderna) och en över Åby älv (undre bilden).

2.4 Accelerometerkalibrering

Äldre mätningar på bron över Långforsen från 2009 hade för dåligt signal-brusförhållande för tillförlitlig modalanalys. Den exakta anledningen till detta var okänd, så flera tänkbara delorsaker åtgärdades. Vi var till exempel noggranna med jordning för att reducera elektriska störningar, elliminera andra tänkbara störkällor vid konstgjord excitering, val av samplingshastighet, filterfrekvenser, ordentlig förankring av accelerometrarna i mätobjekten, genomtänkt utplacering av accelerometrarna (se Figur 2) och ordentlig kalibrering av accelerometrarna. Accelerometerkalibreringen och varför den behövs beskriver vi närmare i detta avsnitt och i Appendix A. I [Bri13] finns en genomgång av en del andra viktiga frågor rörande mätteknik för modalanalys.

Ett enkelt sätt att hitta lämpliga mätpunkter vid vibrationsmätningar för modalanalys är att multiplicera förväntade modformer med varandra och placera accelerometrarna där den

produkten ger stora värden. (Se Figur 2.) Detta ger oftast ett stort antal mätpunkter och även om det går att mäta i omgångar med förlyttning av några accelerometrar mellan varje mätning (se till exempel [Bri13]) så behövs ett flertal accelerometrar. Dessa kan vara av olika fabrikat, det kan vara blandning av enaxlika accelerometrar och accelerometrar som mäter i tre riktningar samtidigt, och mätvärden från olika accelerometrar måste kalibreras så att de gå att jämföra med varandra. Enaxliga accelerometrar kan till exempel kalibreras mot varandra med så kallad *relativ kalibrering*, som i korthet går till så att man placerar alla accelerometrar i "samma" punkt, utför mätning och jämför medelvärden av utsignal från respektive accelerometer för att jämföra hur mycket de förstärker den uppmätta gravitationen. Detta beskrivs närmare exempelvis i [Bri13].

Treaxliga accelerometrar mäter dock i tre olika riktningar samtidigt, och i varje riktning finns ett linjärt samband mellan accelerationen a_n i mätriktningen och accelerometerns utsignal $v_n = a_n g_n + b_n$, där $n = 1, 2, 3$. Parametrarna g_n, b_n är okända och kan variera med temperatur och luftfuktighet, så det är önskvärt att kunna mäta dem ute i fält i den miljö där accelerometern skall användas. Ytterligare ett systematiskt mätfel är elektriskt läckage mellan de tre mätriktningarna, samt liknande "läckage" som tillkommer när de tre mätriktningarna inte är exakt vinkelräta mot varandra. Den kombinerade effekten av de två mätfelen kan beskrivas fullständigt med ytterligare tre parametrar som i frånvaro av elektriskt läckage anger vinklarna mellan de tre mätriktningarna [FGS12].

Det finns olika metoder för att bestämma värdet på dessa parametrar från mätningar av jordgravitationen med accelerometern placerade i vila i sex olika mätpositioner för att bestämma de första sex parametrarna, eller nio olika mätpositioner för att bestämma värdet på alla nio parametrarna. För tidigare föreslagna iterativa algoritmer var det dock oklart under vilka förutsättningar de konvergerar mot korrekta värden på parametrarna. Vi kunde även visa att det finns olika val av sex respektive nio tillsynes lämpliga accelerometerpositioner för vilka det är omöjligt att räkna ut alla sex eller nio parametrar [GS11, FGS12].

För praktiskt användbar accelerometerkalibrering i fält ville vi därför dels ha ett enklare sätt att räkna ut parametrarna, vi ville helst hitta sex respektive nio "säkra" positioner för vilka det garanterat går att räkna ut alla paramterar *även* i fält, där man kanske får räkna med att inte kunna få exakt de positionerna relativt horisontalplanet utan en avvikelse på upp till kanske 10-15 grader. Slutligen ville vi även ha enkla hjälpmedel för att snabbt ställa accelerometern i dessa 6–9 säkra positioner.

För först sex och sedan nio parametrar härledde vi därför explicita formler för att räkna ut parametrarna. Dessa återges som formlerna (2) och (3) i Appendix A, sidan 25.

2.4.1 Sexparameters kalibrering klarar ej läckage mellan axlar

För sex parametrar brukar sex positioner med två axlar i horisontalplanet föreslås. För detta gjorde vi en träkub med numrerade sidor som accelerometrarna snabbt kan skruvas fast på och placeras i sex sådana postitioner, se Figur 14 (c).

För mätningar i fält (på ojämnt ickehorisontellt underlag) behövs lite "vingelmån", en maximal tillåten avvikelse D sådan att för alla möjliga val av sex (eller nio) mätpositioner med upp till D graders avvikelse från de önskade så fås mätningar från vilka det garanterat går att räkna ut korrekta parametervärden. I brusfri miljö, för sex parametrar och med ett generellt antagande som säger ungefär att parametrarna b_1, b_2, b_3 är små jämfört med g_1, g_2, g_3 så kunde vi i [GS11, Avsnitt 2.4] räkna ut en sådan säkerhetsmarginal D. För parametervärden typiska för accelerometrarna i vårt lab så blev D lite drygt 5 grader. En fördel med ett sådant matematiskt bevis är att det garanterar korrekt kalibrering för *alla* tänkbara avvikelser på

upp till D grader för *alla* accelerometrar med parametervärden inom vissa givna intervall, till exempel alla alla accelerometrar av samma modell som vi använder.

En nackdel med ett så generellt bevis är att det dels krävde antagande om brusfri miljö och dels inte ger största möjliga värde på maximala avvikelsen D. För indikation om hur mycket större D som kan utnyttjas i praktiken så gjorde vi även omfattande Monte Carlo-simuleringar för en accelerometer med parametervärden samma som för en av våra accelerometrar och med brusnivåer typiska för de fältmätningar vi gjort [GS11]. Dessa sammanfattas i Appendix A.1, sid 25. De visar att upp till 10 graders avvikelse från de önskade sex positionerna knappt påverkar skattningsfelet i ett idealt fall då accelerometern har perfekt vinkelräta axlar och inget elektriskt läckage mellan mätningarna från dessa. Med sådant läckage av samma storlek som anges i databladen för våra accelerometrar så påverkas dock skattningsfelen betydligt, och nioparameters kalibrering behövs därför för små skattningsfel i verkliga mätningar.

2.4.2 Nioparameters kalibrering bättre för verkliga accelerometrar

För nioparameters kalibrering brukar rekommenderas att utgå från samma sex mätpositioner som för sexparameters kalibrering, samt tre till som fås genom 45 graders vridning från dessa. Det är dock lite vanskligt att välja dessa vid mättillfället, som i Figur 14 (d), då de enklaste valen av 45-gradersvridningar ger mätresultat som ej kan användas för kalibrering (se [FGS12, Avsnitt 3.2]). För att snabbt och enkelt göra korrekta kalibreringsmätningar i fält sågade vi därför av tre vinklelräta sidor av träkuben i Figur 14 (d). Vi satte även i tre skruvar i varje sida för att den skall stå stabilt på ojämna underlag, se Figur 14 (e). Detta var den konstruktionsmässigt enklaste lösningen, men vi förväntade oss bättre resultat om formen ändras till en niosidig polyeder där sidorna väljs för maximal minsta vinkel mellan två mätpositioner, liksom det inte går att minska till mindre än 90 graders vridning mellan två mätpositioner vid sexparameters kalibrering. (Intuitivt kan man se detta som ett sätt att få "så olika" kalibreringsmätningar som möjligt.) Detta ger den polyeder som visas i Figur 15. Med de 3D-skrivare som finns idag så är det relativt enkelt att till exempel konstruera ett nytt accelerometerhölje med denna form.

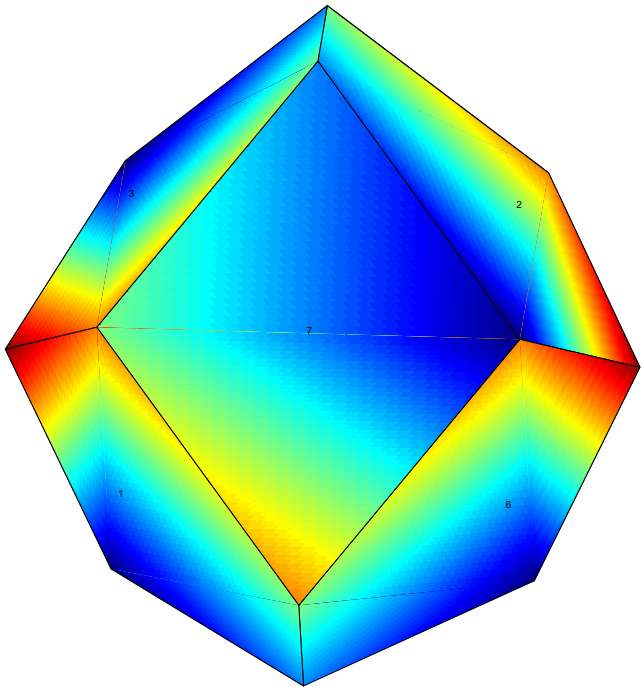
Liksom för sexparameters kalibrering behövs en "vingelmån" för hur många grader man kan avvika från önskade positioner och ändå garanterat kunna bestämma korrekta parametervärden från mätningarna. Även för nioparameters kalibrering gjorde vi ett generellt bevis för brusfri miljö och accelerometrar med parametervärden av samma storleksordning som för de accelero-

Figur 14: Våra treaxliga accelerometrar sitter monterade på en platta (a) med tre skruvar och vattenpass (b) för justering efter horisontalplanet. Träkuben i (c) möjliggör snabb positionering för sexparameters kalibrering. Motsvarande konstruktioner för nioparameters kalibrering visas i (e) och i Figur 15.

metrar vi själva använder. Detta gav dock bara en vingelmån på cirka en halv grad, så de resultaten behöver någon ytterligare förbättring för att vara intressant för publicering. Sådana resultat vore dock intressanta, då de Monte Carlo-simuleringar vi publicerade i [FGS12, Avsnitt 3.2] är begränsade till ett fixt val av parametervärden och mycket tidskrävande (varje simulering som den i Figur 20, sidan 28 tog 9 månader på en bärbar dator med då snabbaste nyaste Intel-processorn). Kompletterande resultat som gäller mer generellt för alla parametervärden av en viss storleksordning vore därför fortfarande intressanta.

De Monte Carlo-simuleringar vi gjort redovisas närmare i [FGS12], där vi använder beteckningen A_0^{90-45} för en mätsetup enligt Figur 14 (e) och $A_0^{\text{max sep}}$ för motsvarande kalibreringsmätningar enligt Figur 15. Simuleringarna är för samma brusnivå som vi uppmätt i våra fältmätningar och samma läckage mellan de tre mätaxlarna som anges i datablad för våra accelerometrar. Simuleringsresultaten som sammanfattas i Appendix A.2 visar att storleken på mätfel efter kalibrering knappt påverkas av slumpartade positioneringsfel på upp till cirka 12 grader för A_0^{90-45} respektive drygt 20 grader för $A_0^{\text{max sep}}$. Denna "vingelmån" är mer än tillräcklig för att få snabba och pålitliga kalibreringsmätningar i fält även på ojämnt underlag med utrustning som i Figur 14 (e) och Figur 15.

Simuleringsresultaten i Appendix A.2 visar också, som väntat, att sexparameters kalibrering ger betydligt större fel i mätresultaten efter kalibrering än man får med nioparameters kalibrering. För nio mätningar med A_0^{90-45}-setup och väldigt höga brusnivåer visar dock simuleringsresultat i [FGS12] att sexparameters kalibrering ger mindre mätfel efter kalibrering, speciellt för stora D.



Figur 15: Niosidig polyeder som skiljer sig från konstruktionen i Figur 14 (e) på så vis att minsta vinkeln mellan två sidor maximerats. Detta ger ger mer vingelmån för kalibreringsmätningar på ojämnt underlag med större avvikelser från de önskade nio positionerna utan att ge märkbart större mätfel efter kalibrering.

3 Diskussion och slutsatser

Mätteknik och metoder för modalanalys och tillståndsanalys av olika konstruktioner har utvecklats, undersökts och utvärderats på en betongplatta i laboratoriemiljö. För betongplattan gav mindre sprickor små ändringar av plattans självsvängningar. För lite större sprickor indikerade preliminära resultat vilken halva av plattan som blivit försvagad. För djupare sprickor gav tillståndsbedömningen mer noggrann lokalisering av skadan.

En förenklad version av MATLAB-koden som användes för tillståndsbedömningen av plattan redovisas i Appendix D.3.27. Vi beskriver där närmare hur FEM-uppdatering med metoden Newton trust region är implementerad. Detta är själva huvudalgoritmen, och den är oberoende av konstruktionens geometri, undantaget om man använder "interpolation functions"-delen av koden, som kan användas för att släta ut lokala variationer i uträknade elastitetsmoduler, men är begränsad till rektangulärt utplacerade nodpunkter.

Vi har även utfört modalanalys på vibrationsmätningar på en flervåningsbyggnad och jämfört med moder och frekvenser förutspådda av en finita elementmodell av byggnaden. För att kunna jämföra förutspådda och uppmätta modformer så måste man ha tillräckligt med mätpunkter på varje våningsplan för att beskriva det planets svängningar. För en flervåningsbyggnad kräver detta betydligt fler mätpunkter än för till exempel en bro. Saknas mätpunkter så kan man välja värden där med linjär interpolation mellan punkter på samma eller intilliggande våningsplan, men då riskerar man samtidigt att missa någon väsentlig detalj i en modform.

För mätningar i fält krävs kalibrering av använda accelerometrar så att mätningar i olika mätpunkter och vid olika mättillfällen blir direkt jämförbara. Vi har utvecklat förenklad analysteknik och enkla hjälpmedel för att vid mätningar i fält snabbt och enkel kalibreria treaxliga accelerometrar mot jordens gravitationskraft och samtidigt reducera mätfel på grund av elektriskt läckage och icke vinkelräta axlar. De resultaten har publicerats som artiklar i internationell tidskrift [GS11, FGS12].

Enklaste formen av det framtagna mäthjälpmedlet är att forma själva höljet till accelerometern som en niosidig polyeder med numrerade sidor enligt Figur 15. Detta möjliggör till exempel batteridrivna accelerometrar för långtidsmätningar som t ex snabbt kan kalibreras om till exempel i samband med batteribyte.

Några frågor av betydelse för större konstruktioner i allmänhet (broar eller ej) redovisas under separata rubriker i avsnitt 4, samt i punktlistan sist i Avsnitt 2.3.1.

4 Fortsatt forskning

4.1 Anpassningar för tillståndsanalys på större konstruktioner

MATLAB-koden som användes för tillståndsanalys på plattan beskrivs närmare i Appendix D. Den körs relativt snabbt på en vanlig bordsdator. För större och mer komplicerade konstruktioner som bron över Åby älv så är FE-modellen avsevärt mycket större, vilket dels kan ge långa exekveringstider och dels skulle kunna ge styvhetsmatriser större än största tillåtna storleken i MATLAB.

Som en motåtgärd planerar vi att utnyttja en datorkluster på 96 kärnor som finns på Institutionen för samhällsbyggnad och naturresurser på Luleå tekniska universitet. Detta visade kräva att både köhanteringssystemet på klustern och MATLABs rutiner för parallell körning på flera kärnor uppdateras för problemfri parallell körning av MATLAB och Abaqus. En lösning är nu på gång, tack vare ett samarbete mellan IT-personal från universitetet, en inhyrd konsult (Patrik Norman, Exait AB) och kanadensiska tekniker från MATLAB.

För att vid behov få ned problemstorleken har vi även gjort en mindre detaljerad finit elementmodell av bron över Åby älv. För att kunna testa tillståndsanalys på andra konstruktioner än betongplattan så behöver MATLAB-koden i Appendix D anpassas. Själva huvudalgoritmen är oberoende av konstruktionens geometri, men framför allt de inledande delar av koden som sätter olika parametervärden ocf konstanter, samt en del rutiner som plottar slutresultat behöver anpassas för andra konstruktioner. För bron över Åby älv har vi upprepat vibrationsmätningar på intakt bro, efter att ha tillfört mindre skador och efter att ha belastat bron till brott. Detta gör den bron till en lämplig första större konstruktion att generalisera koden i Appendix D för, eftersom vi då har en möjlighet att undersöka hur små skador som kan detekteras på samma sätt som gjordes för betongplattan i Figur 9.

4.1.1 Använda derivator av målfunktionen eller ej?

Koden i Appendix D mäter "avståndet" mellan förutspådda och uppmätta moddata (modformer och -frekvenser) med en så kallad *målfunktion* (*objective function*). Ett lokalt minimum till målfunktionen söks med en optimiseringsmetod (Newton Trust Region [NW06, Chapter 4]) som utnyttjar målfunktionens värde, derivator och andraderivator för att stegvis justera alla nyckelparametrar i lämplig riktning. Skulle det vara svårt att få konvergens för större konstruktioner så kan man även prova att använda liknande metoder som ej försöker utnyttja både värde, derivata och andraderivata. Detta gjordes till exempel vid en studie på Svinesundsbron [SPG09], där konvergensproblem för en gradientbaserad metod åtgärdades genom att istället använda en motsvarande algoritm som ej utnyttjar gradienten. Vi har som komplement även implementerat en metod [MLF11] (se appendix D.1 och D.3.24), som enbart utnyttjar förstaderivatorna och kan vid behov även implementera samma metod som i [SPG09] eller liknande ifall koden i Appendix D skulle ge konvergensproblem för någon större konstruktioner.

4.2 Mer generell vingelmån för nioparameters kalibrering

Som beskrivits i Avsnitt 2.4.2, så är de Monte Carlo-simuleringar som gjort extremt tidskrävande och begränsade till parameteruppsättning för en bestämd accelerometer, så det vore värdefullt att även för nioparameters kalibrering kompletera med teoretiska resultat som garanterar en lägre vingelmån, men generellt för alla accelerometrar med parametervärden inom något visst spann.

4.3 Kontinuerliga mätningar

För de mätningar i forskningssyfte som beskrivits hittills så har varje enskild mätning krävt rätt stora personella, tidsmässiga och ekonomiska resurser. Detta ger möjlighet att vid varje nytt tillfälle göra olika optimeringar av mätproceduren för exakt den typ av analys man vill utföra.

På lång sikt är det önskvärt att vid underhåll eller redan vid byggandet av konstruktionen applicera in accelerometrar eller andra sensorer för kontinuerlig insamling av mätdata för kontinuerlig trendanalys av hur olika nyckelparametrar förändras med tiden för att på ett tidigt stadium kunna varna för behov av underhåll. Används batteridrivna accelerometrar som skickar mätdata trådlöst så underlättas även kontinuerliga mätningar på konstruktioner långt från annan bebyggelse. Då går kanske inte att flytta omkring och kalibrera om accelerometrar mellan mättillfällen, men i gengäld kan det finnas ett stort överflöd av mätdata. Detta kan användas till att reducera signal-brusförhållande, men samtidigt ställer det större krav på att kunna lagra och effektivt analysera väldigt stora datamängder. Med hölje format som i Figur 15 kan nya

kalibreringsmätningar snabbt genomföras till exempel i samband med batteribyte. Eftersom kalibreringen elliminerar mätfel på grund av ickeortogonala axlar respektive elektriskt läckage mellan mätriktingarna så ger detta även lägre krav på hårdvaran.

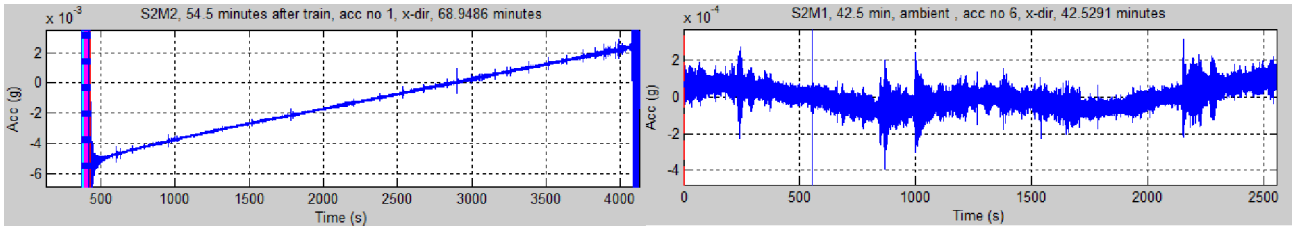
Niklas Grip undersöker just nu sådana möjligheter ihop med Fredrik Sandin på EISLAB vid Luleå Tekniska universitet. EISLAB är bra just på batteridrivna sensorer som skickar data via mobiltelefonnätet, och Fredrik har intresse för vissa tekniker för representation som kan vara intressanta att jämföra med till exempel waveletbaserade tekniker som Niklas är expert på.

4.4 Frekvensupplösning

Svängningsmoder som ligger nära i frekvens, är vanligt förekommande speciellt i "tornformade" byggnationer, men förekommer även i våra mätningar på bron över Långforsen. Då vi aldrig sett någon jämföra de påstått höga frekvensupplösningarna hos FDD och den beskrivna waveletmetoden i Avsnitt 2.1 så är detta en intressant fråga att undersöka närmare.

4.5 Korrigering av drivande mätvärden

Med den utrustning vi har så skall egentligen den uppmätta utsignalen har en nollnivå (lokalt medelvärde) som håller sig konstant, men under långa mätningar så kan det förekomma tydliga långsamma tidsvariationer hos nollnivån, som i Figur 16. Bäst vore förstås om man kan hitta och eliminera orsaken till drivande nollnivå, men för mätningar där det redan inträffat vore det bra att kunna korrigera i efterhand. Oftast ser det ut som i vänstra figuren, och då är en enkel lösning att minstakvadrat-anpassa till en rät linje och subtrahera bort den linjen. Mer sällan kan det vara som i den högra figuren, där nollnivån varierar med en tidsskala (våglängd) på ungefär 500 sekunder. Vi har än så länge klarat oss utan att använda sådana signaler, men ett möjlig sätt att korrigera är att subtrahera bort önskade långsama variationer med samma waveletbaserade teknik som används i [GGN13].



Figur 16: Två exempel på tidssignal där "nollnivån" driver. (Från mätningar på bron över Åby älv 2012.)

APPENDIX

A Accelerometerkalibrering

Accelerometerkalibrering behövs för att kunna räkna om accelerometerns utsignal (till exempel spänning i enheten volt) till motsvarande värden på den uppmätta accelerationen, till exempel i enhet antal g. Vi beskriver här detta samband samt hur kalibreringen går till. Mer detaljerad härledning, analys och diskussioner finns i [GS11, FGS12]. Vi använder enkel matrisnotation som gör det lätt att snabbt utföra motsvarande beräkningar i till exempel MATLAB.

Utsignalen från en treaxlig accelerometer ordnas i en matris

$$\mathcal{V} = \begin{pmatrix} V_{\mathrm{x},1} & V_{\mathrm{x},2} & V_{\mathrm{x},3} & \cdots & V_{\mathrm{x},N} \\ V_{\mathrm{y},1} & V_{\mathrm{y},2} & V_{\mathrm{y},3} & \cdots & V_{\mathrm{y},N} \\ V_{\mathrm{z},1} & V_{\mathrm{z},2} & V_{\mathrm{z},3} & \cdots & V_{\mathrm{z},N} \end{pmatrix}.$$

Med hjälp av nio parametrar α_m, b_m, g_m kan motsvarande värden för den verkliga acceleration skrivas som matrisen

$$\mathcal{A} = F(\mathcal{V} - B), \quad \text{med} \quad F = \begin{pmatrix} \frac{1}{g_1} & 0 & 0 \\ \frac{\alpha_3}{g_1} & \frac{1}{g_2} & 0 \\ -\frac{\alpha_2}{g_1} & \frac{\alpha_1}{g_2} & \frac{1}{g_3} \end{pmatrix} \quad \text{och} \quad B \overset{\text{def}}{=} \underbrace{\begin{pmatrix} b_1 & b_1 & \dots & b_1 \\ b_2 & b_2 & \dots & b_2 \\ b_3 & b_3 & \dots & b_3 \end{pmatrix}}_{N \text{ kolumner}}. \tag{1}$$

Sambandet (1) beror enbart av sex parametrer i specialfallet $\alpha_1 = \alpha_2 = \alpha_3 = 0$, som motsvarar en accelerometer med perfekt vinkelräta axlar och ingen elektriskt läckage mellan de tre mätriktningarna. Parametrarna kan mätas och räknas ut på följande sätt:

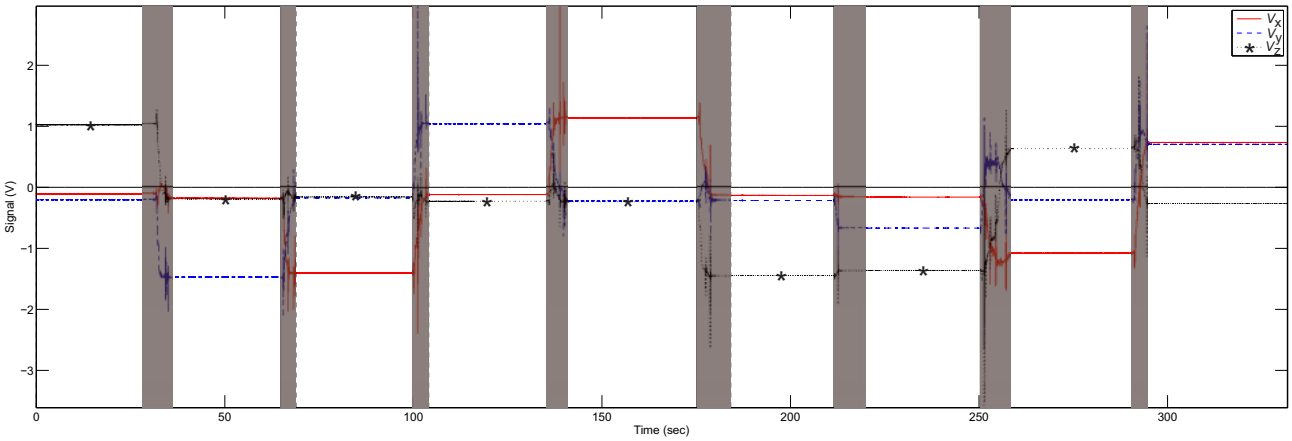
1. Gör mätningar med accelerometern i vila i sex respektive nio olika positioner (samma som antalet skattade parametrar).

2. För brusreducering räknas tidsmedelvärde ut för vart och ett av de sex eller nio tidsintervallen. Se Figur 17.

3. Placera dessa medelvärden i en 3×6- eller 3×9-matris V och gör sedan en av följande:

   Sexparameters kalibrering: Bilda matrisen

   $$V^+ \overset{\text{def}}{=} \begin{bmatrix} P_2(V^{\mathrm{T}}) & V^{\mathrm{T}} \end{bmatrix} \qquad \text{med elementvis kvadrering} \qquad (P_2(M))_{m,n} \overset{\text{def}}{=} M_{m,n}^2.$$



Figur 17: Mätdata för de tre mätriktningarna med accelerometern placerad i vila i nio olika positioner. De gråmarkerade områdena tas bort från mätningarna och för varje kvarvarande tidsintervall räknas tre tidsmedelvärden ut.

Räkna ut $\boldsymbol{u} \stackrel{\text{def}}{=} [u_1\ u_2\ u_3\ w_1\ w_2\ w_3]^T$ sådant att $V^+\boldsymbol{u} = \mathbf{1}$. De sökta parametrarna är

$$b_m = -\frac{w_m}{2u_m} \qquad \text{och} \qquad g_m = \sqrt{\frac{1 + \sum_{k=1}^{3} \frac{w_k^2}{4u_k}}{u_m}}. \tag{2}$$

Nioparameters kalibrering: Låt V^+ vara 9×9-matrisen med $n$te raden $[V_{1,n}^2\ \ V_{2,n}^2\ \ V_{3,n}^2\ \ V_{1,n}V_{2,n}\ \ V_{1,n}V_{3,n}\ \ V_{2,n}V_{3,n}\ \ V_{1,n}\ \ V_{2,n}\ \ V_{3,n}]$. Lös ekvationen $V^+\boldsymbol{u} = \mathbf{1}$. För matrisen

$$U \stackrel{\text{def}}{=} \begin{pmatrix} 2u_1 & u_4 & u_5 \\ u_4 & 2u_2 & u_6 \\ u_5 & u_6 & 2u_3 \end{pmatrix} \qquad \text{sätt} \qquad \Omega' \stackrel{\text{def}}{=} 1 + \tfrac{1}{2}\begin{pmatrix} u_7 & u_8 & u_9 \end{pmatrix} U^{-1} \begin{pmatrix} u_7 \\ u_8 \\ u_9 \end{pmatrix}.$$

De sökta parametrarna α_m, b_m, g_m är

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = -U^{-1} \begin{bmatrix} u_7 \\ u_8 \\ u_9 \end{bmatrix} \tag{3a}$$

$$g_1 = \sqrt{\tfrac{2(4u_2u_3 - {u_6}^2)\Omega'}{\det(U)}}, \qquad g_2 = 2\sqrt{\tfrac{u_3\Omega'}{4u_2u_3 - {u_6}^2}}, \qquad g_3 = \sqrt{\tfrac{\Omega'}{u_3}}, \tag{3b}$$

$$\alpha_1 = \tfrac{u_6 \operatorname{sgn}(\Omega')}{\sqrt{4u_2u_3 - {u_6}^2}}, \qquad \alpha_2 = -\tfrac{u_5\sqrt{4u_2u_3 - {u_6}^2}\operatorname{sgn}(\Omega')}{\sqrt{2u_3 \det(U)}} \qquad \text{och} \qquad \alpha_3 = \tfrac{2u_3u_4 - u_5u_6}{\sqrt{2u_3 \det(U)}}. \tag{3c}$$

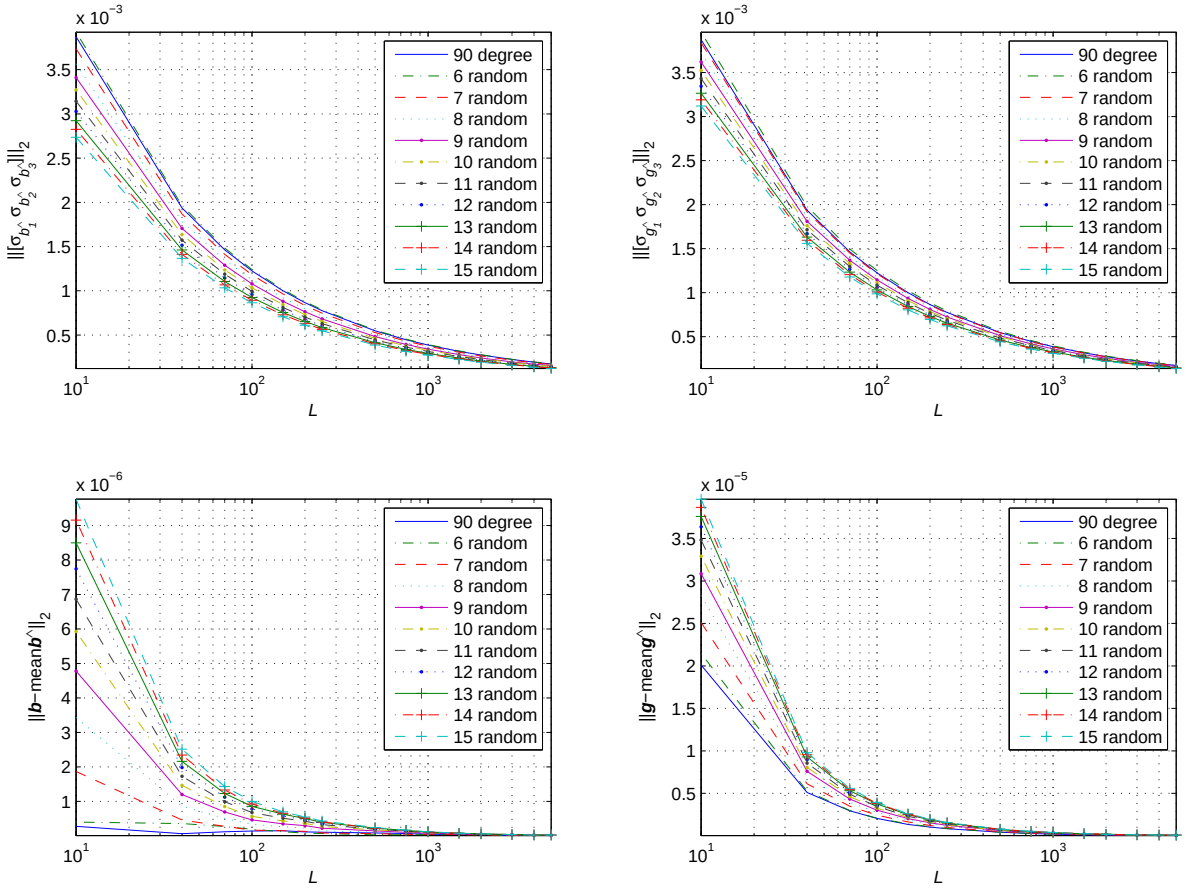
A.1 Monte Carlo-simuleringar för sexparameters kalibrering

För en accelerometer med parametervärden b_g, g_m samma som för en av våra accelerometrar, men perfekt vinkelräta mätriktningar och inget elektriskt läckage mellan dessa har vi gjort omfattande Monte Carlo-simuleringar. Vi simulerade med brusnivåer samma som Figur 18 visar medelvärde och spridning (standardavvikelse) för felet i de uträknade parametrarna. För varje position av accelerometern tas medelvärde av L sampelvärden (jämför Figur 17). Större L motsvarar alltså lägre brusnivå efter medelvärdestagning. De heldragna kurvorna ("90 degrees") i Figur 18 visar hur felet i parameterskattningen avtar med ökande L för ett idealt fall då accelerometern placeras exakt i de sex önskade positionerna med två av mätaxlarna i horisontalplanet. De punktstreckade kurvorna ("6 random") visar motsvarande resultat för en mer realistisk simulering, där varje mätning gjordes med en slumpartad avvikelse med upp till 10 grader från önskad position, vilket är ungefär vad man kan vänta sig vid snabba kalibreringsmätningar i fält, till exempel med utrustningen i Figur 14. Skattningsfelet blir då bara marginellt större. Övriga plottade kurvor visar att om man upp till 10 graders positioneringsfel D och offrar mer tid på att göra fler mätningar (totalt 7–15 stycken) med accelerometern ställd i ytterligare slumpvis valda positioner så kan felet reduceras ytterligare. (Kalibreringsräkningarna modifieras då till en minstakvadratlösning av ekvationen $V^+\boldsymbol{u} = \mathbf{1}$, se [GS11].)

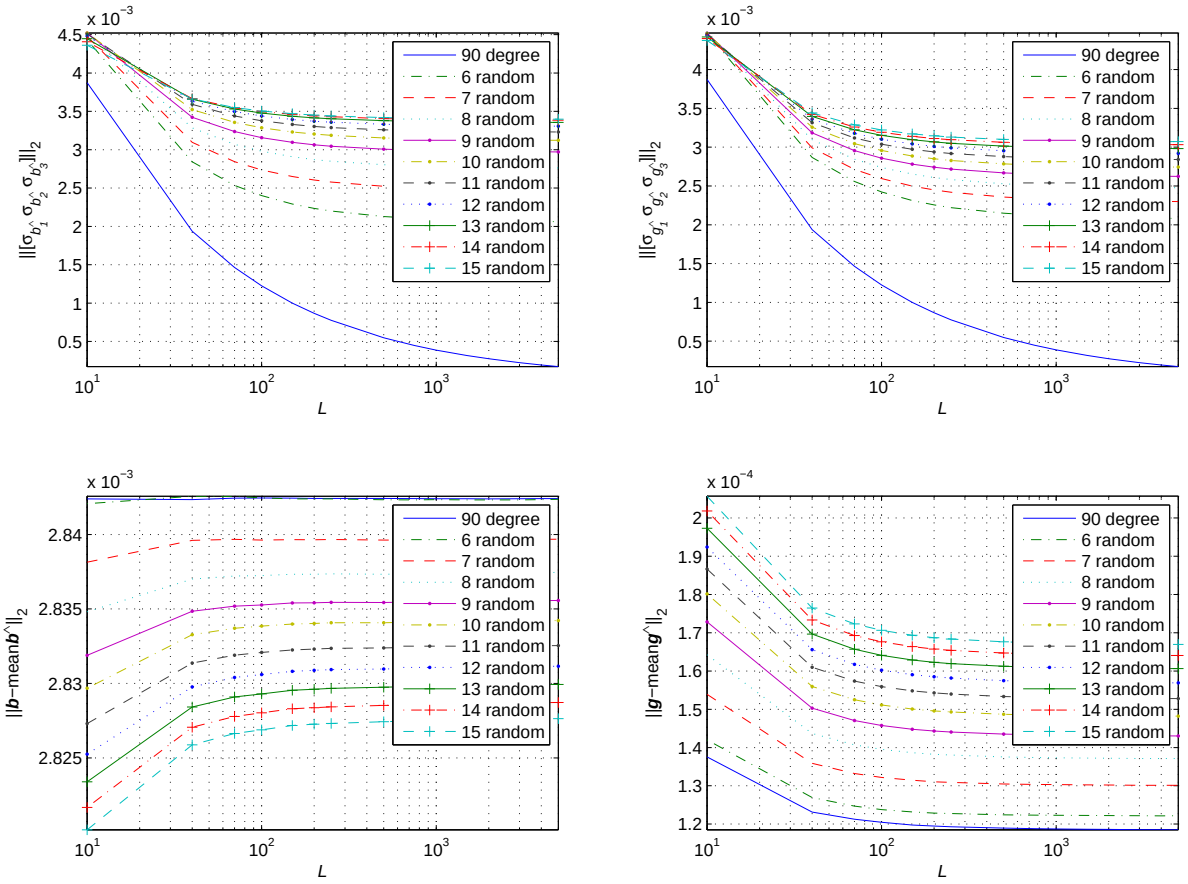
Verkliga accelerometrar har dock elektriskt läckage mellan de olika mätriktningarna. Detta anges till 0.5 % i databladen för de accelerometrar vi själva avnvänder. Lägger man in samma läckage i simuleringarna så får kurvorna i Figur 19. Man ser där att medelvärden över större antal sampel L inte längre reducerar estimeringsfelet lika mycket. Istället behövs nioparameters kalibrering, där de tre extra parametrarna beskriver just den kombinerade effekten av elektriskt läckage och motsvarande läckage på grund av icke perfekt vinkelräta axlar.

A.2 Monte Carlo-simuleringar för nioparameters kalibrering

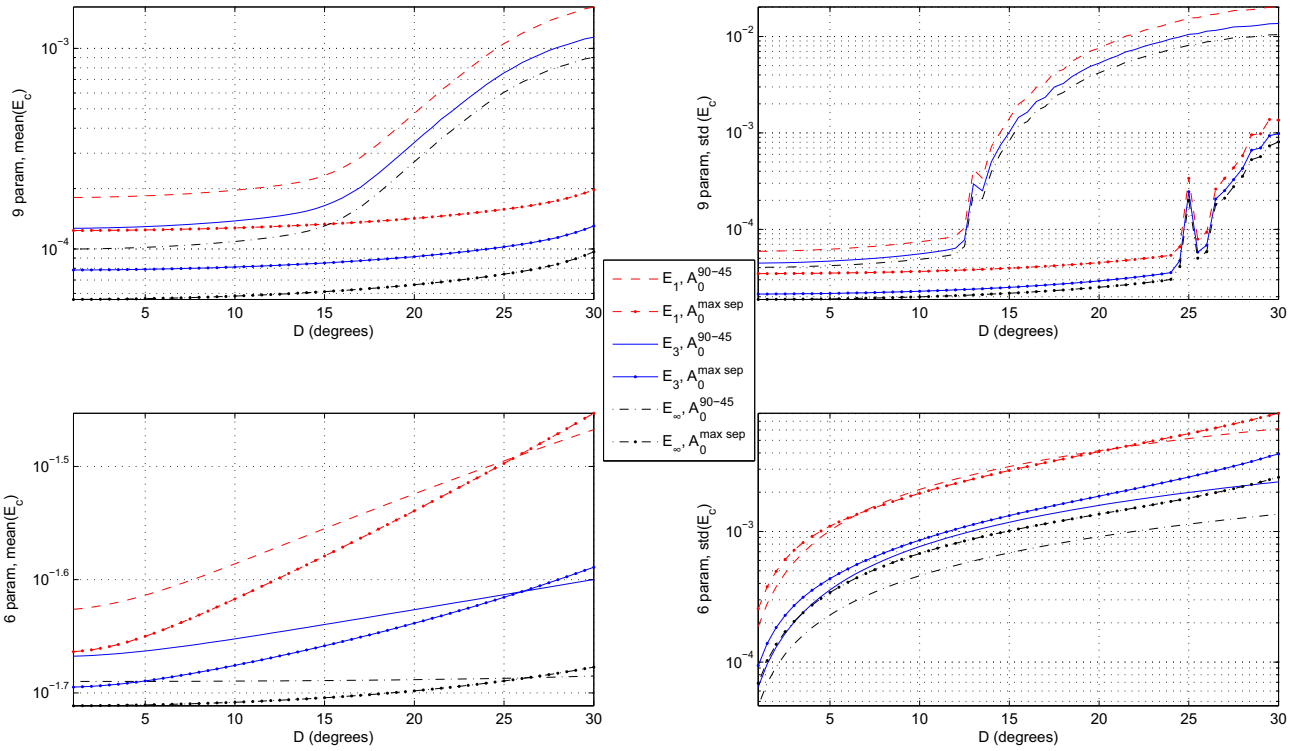
För nioparameters kalibreringar har vi gjort omfattande simuleringar som beskrivs mer i detalj i [FGS12]. Vi väljer där att plotta ett mått på hur stort relativa mätfelet blir efter accelerometerkalibrering. Vi kallar det mätfelet E_3 för mätning av acceleration nära 1 g (vanligt vid accelerationsmätningar på stillastående konstruktioner) och plottar även två motsvarande relativa fel, E_1 för mätning av mindre accelerationer och E_∞ för mätning av extremt stora accelerationer. Se [FGS12] för detaljer. Figur 20 visar att nioparameters kalibrering ger betydligt mindre mätfel efter kalibrering. För varianten med 90- och 45-gradersvridningar, som i Figur 14 (e), så ger slumpvisa avvikelser från dessa önskade positioner med upp till 13 grader marginell påverkan på mätfelet. Med motsvarande mätpositioner enligt Figur 15 så ökar den vingelmånen till ca 23 grader och mätfelen efter kalibrering blir aningen mindre.



Figur 18: Medelvärde och spridning (standardavvikelse σ) för estimeringsfelet i skattning av parametrarna b_m, g_m. Resultat av Monte Carlo simulering (127 300 000 iterationer) med samma brusnivå som i verkliga mätningar, men idealiserat fall med perfekt vinkelräta mätaxlar och inget elektriskt läckage mellan mätriktningarna. Genom att mäta längre i varje position ta medelvärde av ett större antal sampelvärden L, så reduceras brusnivån och estimeringsfelen.



Figur 19: Samma som Figur 18, men nu med elektriskt läckage mellan mätriktningarna. (44 750 000 Monte Carlo-iterationer.)



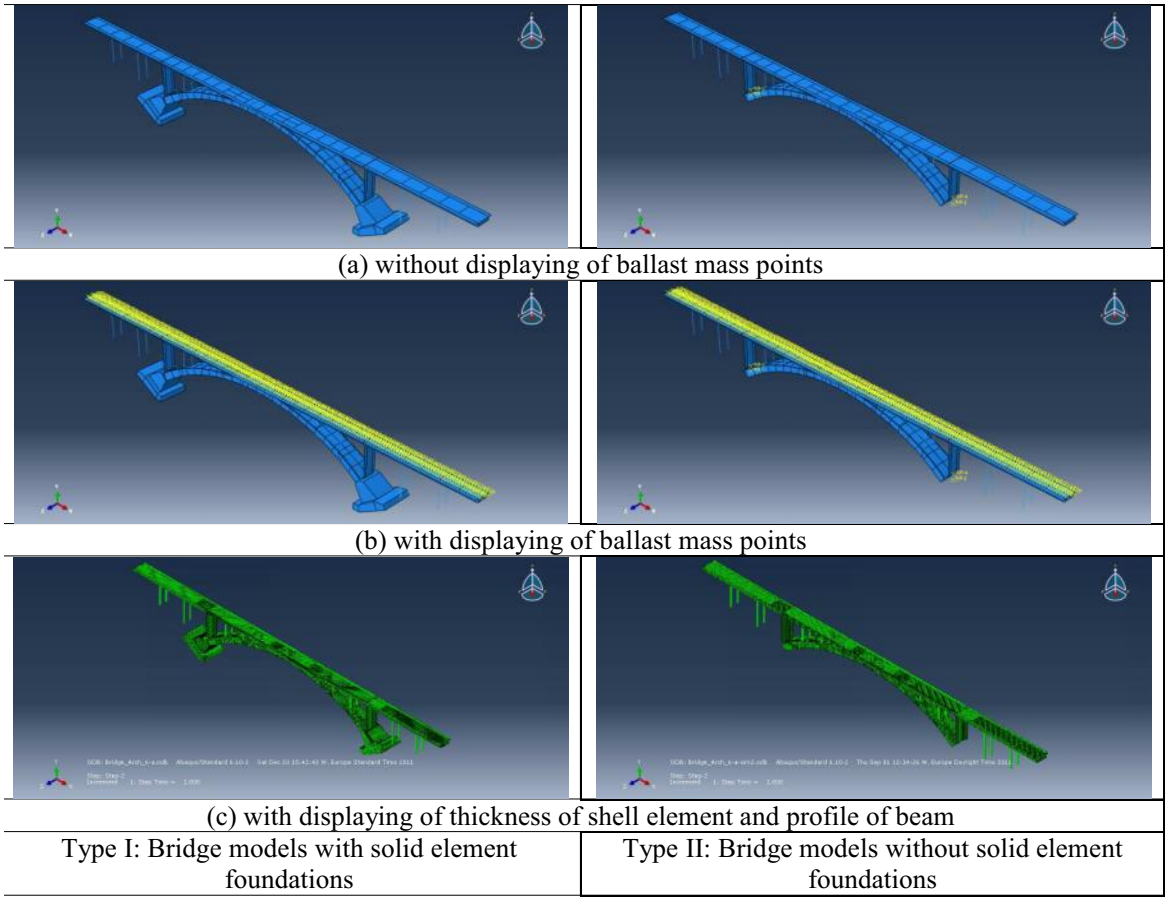
Figur 20: Relativa mätfelet för en accelerometer efter kalibrering, E_3 för accelerationer nära 1 g, E_1 för mindre och E_∞ för mätning av extremt stora accelerationer. Resultat från Monte Carlo-simulering (114 892 520 iterationer) med parametrar och läckage mellan mätriktningar typiska för våra egna Colibrys SF3000L accelerometrar, samt brusnivå typisk för verkliga mätningar i fält.
Övre plottarna visar medelvärde (vänster) och spridning eller standardavvikelse (höger) för mätfelet efter nioparameters kalibrering. Nedre plottarna visar motsvarande för sex parametrar. Mätfelen blir flera tiopotenser mindre för nioparameters kalibrering. Med mätutrustningen i Figur 14 (e) fås väldigt små ändringar av mätfelet vid slumpvisa avvikelser upp till ca 12 grader från de önskade positionerna. Motsvarande för mätpositioner $A_0^{\text{max sep}}$ som ges av mätverktyg format som i Figur 15 ökar den "vingelmånen" från 12 upp till circa 23 grader och ger generellt mnågot mindre mätfel.

B Bromätningar

B.1 Bron över Långforsen

Järnvägsbron över Långforsen utanför Kalix är en 177 meter lång och 60 meter hög betongbro som byggdes 1960 (se Figur 13). Då brons ägare Trafikverket ville öka tillåten hastighet samt öka högsta tillåtna axellasten från 225 till 300 kN, så utfördes 2011–2012 vibrationsmätningar samt mätningar av nedböjning och töjning under tågpassage vid olika hastgheter. Det var oklart om det var tillräckligt med vind för att excitera fria vibrationer starka nog för modalanalys. Som helgardering så lät vi därför ett lok passera precis före varje mätning (se övre högra bilden i Figur 13, sidan 16). Själva lokpassagen kan ej ingå i de analyserade mätningarna, då till exempel slamrande hjul mot räls och broändar mot fundament kan ge olika ickelinjära effekter och ett annorlunda fysikaliskt system än de finita elementmodeller som mätningarna skall jämföras mot.

Vinden avtog drastiskt under mätdagarna, vilket ger gradvis sämre signal-brusförhållande, men fortfarande tillräckligt för att ge mycket bättre överensstämmelse mellan uppmätta och svängningsmoder än vi kunde få med mätningarna från 2009. Vi sammanfattar arbetet och några huvudresultat här. En mer övergripande beskrivning finns bland annat i konferensbidraget [SGP+12] (bifogat som Appendix C).



Figur 21: Översikt över de två bromodellerna. Fundamenten för bropelarna har en enklare modellering med fjädrar i den enklare modellen till höger.

Finita elementmodeller

Gästprofessor Yongming Tu gjorde två finita elementmodeller av bron i Abaqus/Brigade. En som inkluderar detaljerad modell av fundamenten för bropelarna, och en enklare modell där fjädrar används för att modellera randvillkoren vid de fundamenten. Se Figur 21.

Den förstnämnda modellen är mer lik den verkliga konstruktionen så de vibrationsmoder som förutsägs av den modellen kan förväntas ligga närmare uppmätta vibrationer, men kostnaden för detta är en större och mer tidskrävande modell, totalt 93 910 element och 438 800 variabler, vilket reduceras till 47 438 element och 282 808 variabler med den enklare modellen.

Analys och slutsatser

Från uppmätta vibrationer har brons svängningsmoder och frekveser räknats ut med metoderna *frequency domain decomposition* [BZA01, BVA01] och *stochastic subspace identification* [PDR01, OM96] i programmet ARTeMIS. Manuell uppdatering av finita elementmodellerna gjordes genom jämförelser av uppmätta och förutspådda svängningsmoder, töjning och nedböjningar.

Båda finita elementmodellerna ger efter detta egenfrekvenser och nedböjningar nära de som uppmättes vid verkliga tågpassager. En jämförelse gjordes av några av de uppmätta och föutspådda töjningarna i en sektion av de övre balkarna som bär upp plattan och rälsen nära bågens mittsektion. Den delen av bron hade varit den kritiska vid tidigare tillståndsbedömningar av bron. Största uppmätta och förutspådda töjningen under passage av ett lok med axellast 180 kN var nära varandra och av storleksordningen $\Delta\epsilon = 20 \cdot 10^{-6}$, motsvarande ett tryck av storleksordning $\Delta\sigma = 4$ MPa.

Metoder för att bedöma hur stora laster en bro kan bära har utvecklats i EU-projektet Sustainable Bridges [Sus08, HZE+08], och de metoderna har använts här. Tidigare undersökningar hade visat att bron klarade en axellast på 330 kN. De enda kritiska delarna var lokaliserade i plattan på bågens topp: (1) vid själva kontaktytan med bågen där det är stora spänningar i de översta armeringarna ; och (2) i transversell riktining (sidled) mellan de stödjande pelarna, där de undre armeringarna utsätts för starka spänningar. Vi har undersökt dessa delarna i finita elementmodellen för olika laster och preliminära resultat gav låga spänningar av storleksordningen 10 MPa de kritiska delarna och vertikala nedböjningar vid bågens topp av storleksordning ±5 mm, vilket indikerar att bron klarar en axellast 330 kN utan problem.

B.2 Bron över Åby älv

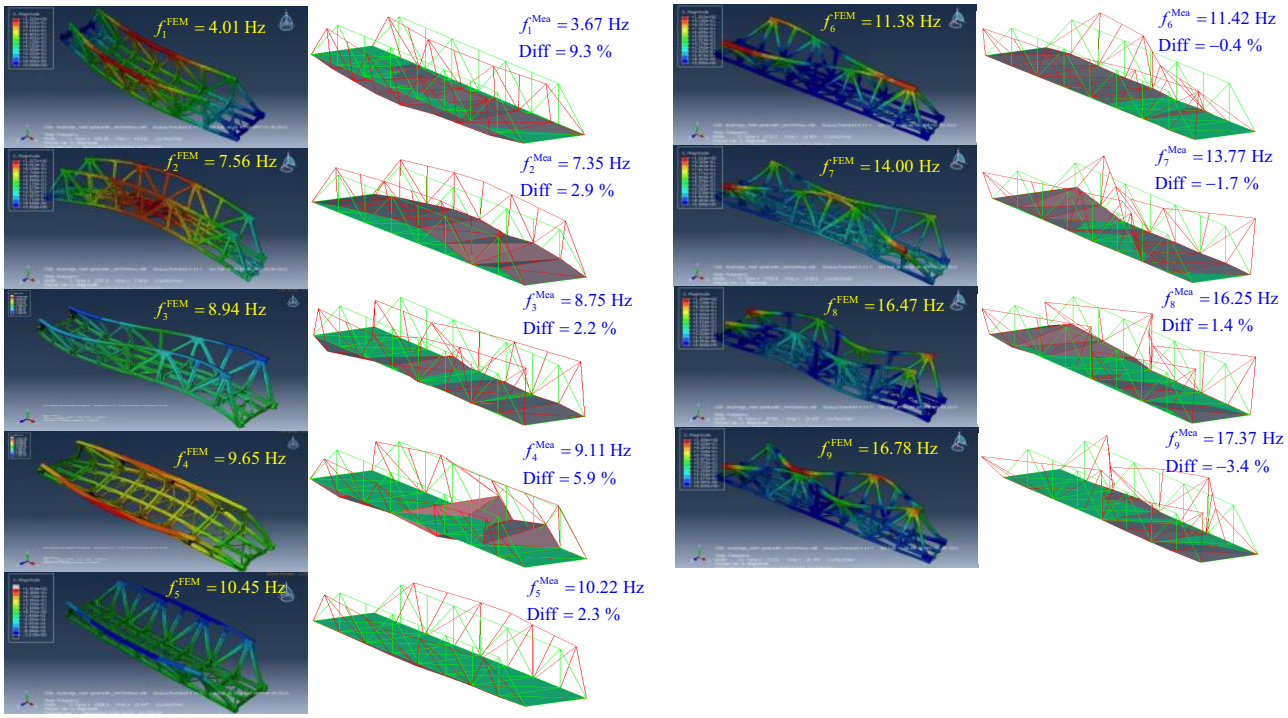
Då trafikverket under 2012 skulle byta ut en järnvägsbro över Åby älv så gjordes 2012–2013 en serie tester och mätningar som kommer att redovisas närmare bland annat i EU-projeket MAINLINE [Mai11]. Första serien vibrationsmätningar gjordes September 2012 medan bron fortfarande var i bruk (Figur 13, sidan 16). Accelerometrarna placerades då i totalt 10 mätpunkter på övre balkarna och 8 på undre längsgående balkar. På grund av närhet till högspänningsledningar så fick accelerometrarna ej placeras ovanpå de övre balkarna, utan fästes med tvingar på insidan av balkarna. Det var osäkert om enbart vind och älv kunde excitera tillräckligt med fria svängningar för modalanalys, så varje mätning påbörjades direkt efter en tågpassage för att även tillföra svängningar exciterade av tågpassage i början av varje mätning. Signalerna såg relativt brusiga ut vid plot av spektrum, men nio vibrationsmoder kunde identifieras och dessa jämfördes mot en noggrann finita element modell av bron som gästprofessor Yongming Tu utvecklat. Frekvenserna skilde som mest 9,3 % för första moden, 5,9 % för den fjärde och låg mellan 0,4 % och 2,9 % för de övriga. Se jämförelse av modformer och frekvenser i Figur 22 för

den första mätomgången då bron fortfarande var i bruk och fria vibrationer efter tågpassager mättes.

När den nya bron installerades placerades den gamla på land med samma typ av infästning av broändarna som tidigare. Nya mätningarna gjordes nu i samarbete med forskare från Avdelningen för bro- och stålbyggnad vid KTH under ledning av Dr. Andreas Andersson, samt en polsk forskargrupp ledd av professor Jarosaw Zwolski vid Wroclaw tekniska universitet i Polen. De hade med sig en shaker som användes för excitering av bron. Första testkörningen avslöjade lösa fästen som gjorde att rälsen slog mot resten av bron samtidigt som även gallerdurk och några mindre balkar längs sidorna slamrade. Detta ger okända insignaler där räls, balkar, gallerdurk och bro slår mot varandra, men för excitering med shaker är det viktigt att shakern är den enda insignalen av betydelse och att den insignalen mäts och kan använda i efterföljande modalanalys av brons frekvenssvar (se [CJK06, Avsnitt 18.6.1]). Därför svetsades rälsen fast kring shakern och lös gallerdurk och småbalkar monterades bort, som visas i Figur 23.

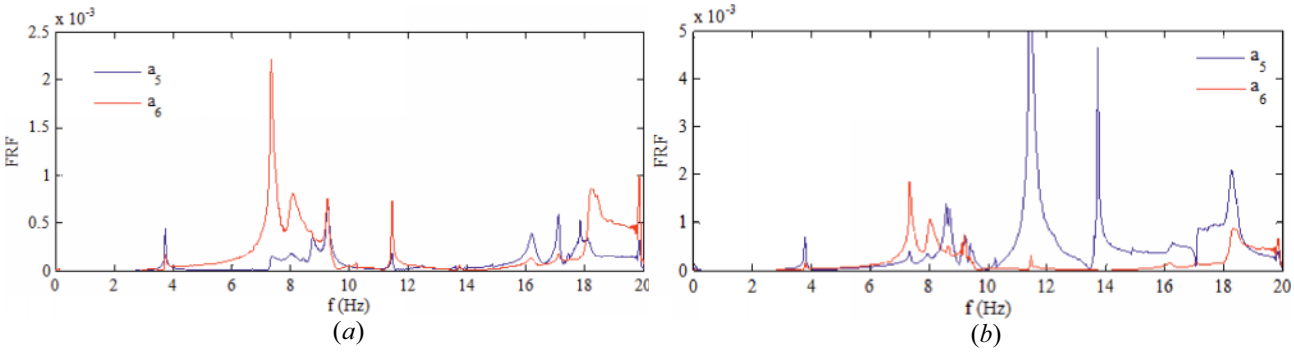
Efter detta fungerade exciteringen som önskat, och vibrationsmätningar tog vid med treaxliga accelerometrar från Luleå, enaxliga från polen, samt en- och tvåaxliga från KTH, se Figur 23. Individuell kalibrering av de olika accelerometrarna (enligt avsnitt 2.4 och Appendix A för de treaxliga) säkerställer dock att få mätdata i samma enhet (antal g) från samtliga så att de olika mätningarna kan kombineras.

De nya vibrationsmätningar gjordes med accelerometrarna placerade i 41 stycken mätpunkter enligt Figur 25. För varje mätuppställning kördes en cyklisk belastning med exciteraren där frekvensen varierades kontinuerligt från 3-20 Hz under 25 min. Data samlades in med en samplingsfrekvens på 800 Hz. Fullständig modalanalys är ännu ej utförd, men Figur 24 visar frekvensssvaret (Frequency Response Function) för två mätpunkter i närheten av exciteraren. En



Figur 22: För bron över Åby älv har modformer och frekvenser förutspåtts av FEM-modell. Dessa jämförs här mot de som fås ur modalanalys i ARTeMIS av uppmätta fria vibrationer medan bron var i bruk. Differensen i mellan uppmätta och förutspådda modfrekvenser ligger under 3,4 % för alla modsvängningar utom två.

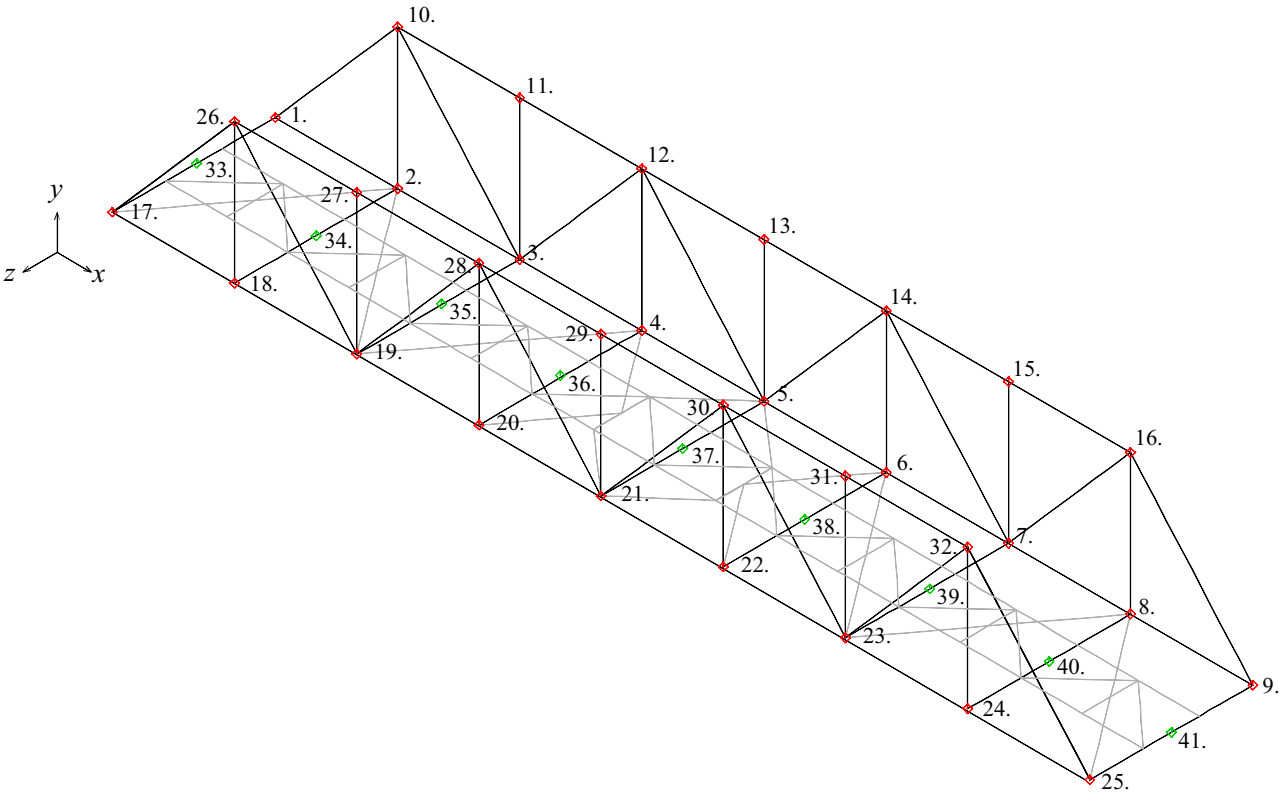
Figur 23: *Bild 1–5:* Vid mätpunkt 35 i Figur 25 placerades den polska forskargruppens shaker. Lösa fästen för rälsen ersattes med svetsfogar. Skallrande gallerdurk och småbalkar plockades bort.
Bild 6–7: Vibrationsmätningar gjordes med accelerometrar från både Luleå, KTH och Polen. De från Luleå är treaxliga och kalibrerades med den metod som beskrivs i avsnitt 2.4 och Appendix A.



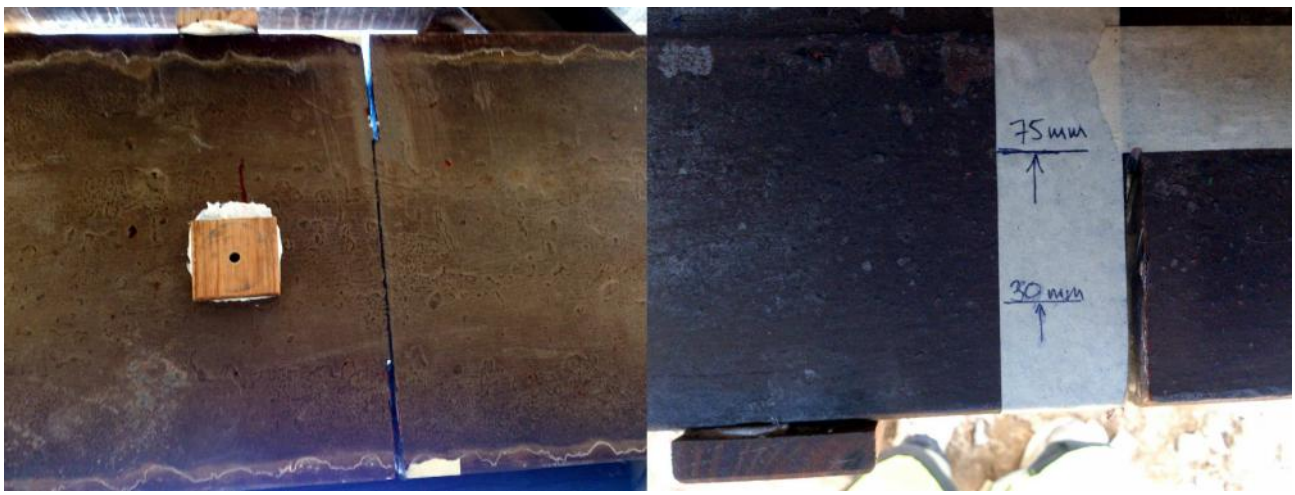
Figur 24: (a) FRF i horisontell (a5) och lodrät (a6) riktning för mätpunkt 3 i Figur 25.
(b) FRF i horisontell (a5) och lodrät (a6) riktning för mätpunkt 11 i Figur 25.

närmare studie i Figur 24 (a) visar moder ungefär vid 3,7 Hz, 7,4 Hz, 8,1 Hz, 8,7 Hz, 9,3 Hz, 11,4 Hz, 16,2 Hz och 17,3 Hz. Uppskattad dämpning för första och andra moden är 0,4 % och 0,6 % baserat på Half-Power Bandwidth metoden och kurvanpassning i frekvensdomän. Motsvarande studie i Figur 24 (b) visar moder vid ungefär samma frekvenser, till exempel vid 3,8 Hz för första moden.

Samma mätningar upprepades sedan två gånger efter att ha tillfört två nivåer av mindre konstgjord skada mitt på den undre tvärgående balken vid mätpunkt 35, se Figur 26. Bron be-



Figur 25: Mätpunkter för bron över Åby älv.



Figur 26: Två små skador med djup 30 respektive 75 mm tillfördes vid mätpunkt 35 i Figur 25.

lastades slutligen till brott, se Figur 27. Efter detta gjordes en sista omgång vibrationsmätningar på 20 av mätpunkterna i Figur 25. Vid dessa mätningar fanns ingen shaker kvar på plats, så nu mättes enbart fria vibrationer, exciterade av dels vind (hyfsat stark på morgonen men svag på kvällen) och förbipasserande tåg. Då tågen passerade på järnvägen intill bidrog de med vibrationer som var tillräckligt starka för att kännas i marken och excitera bron, men inte så starka att de bedömdes kunna störa mätningarna av de fria vibrationerna. Därmed kunde två till tre tågpassager per mätning ingå. Vid något mättillfälle så provades även manuell excitering med slumpvisa hammarslag runtom på bron sista 5-10 minuterna. Analysresultaten blev dock sämre om hammarslagen inkluderades, och det var bra signal-brusförhållande även utan dessa. Modalanalys med ARTeMIS gav modformer och frekvenser som visas i Figur 28 och 29. Inom parantes anges där nummer för motsvarande vibrationsmoder i motsvarande modform i Figur 22. Tabell 1 jämför modfrekvenserna i de tre figurerna. För att se huruvida ändringar

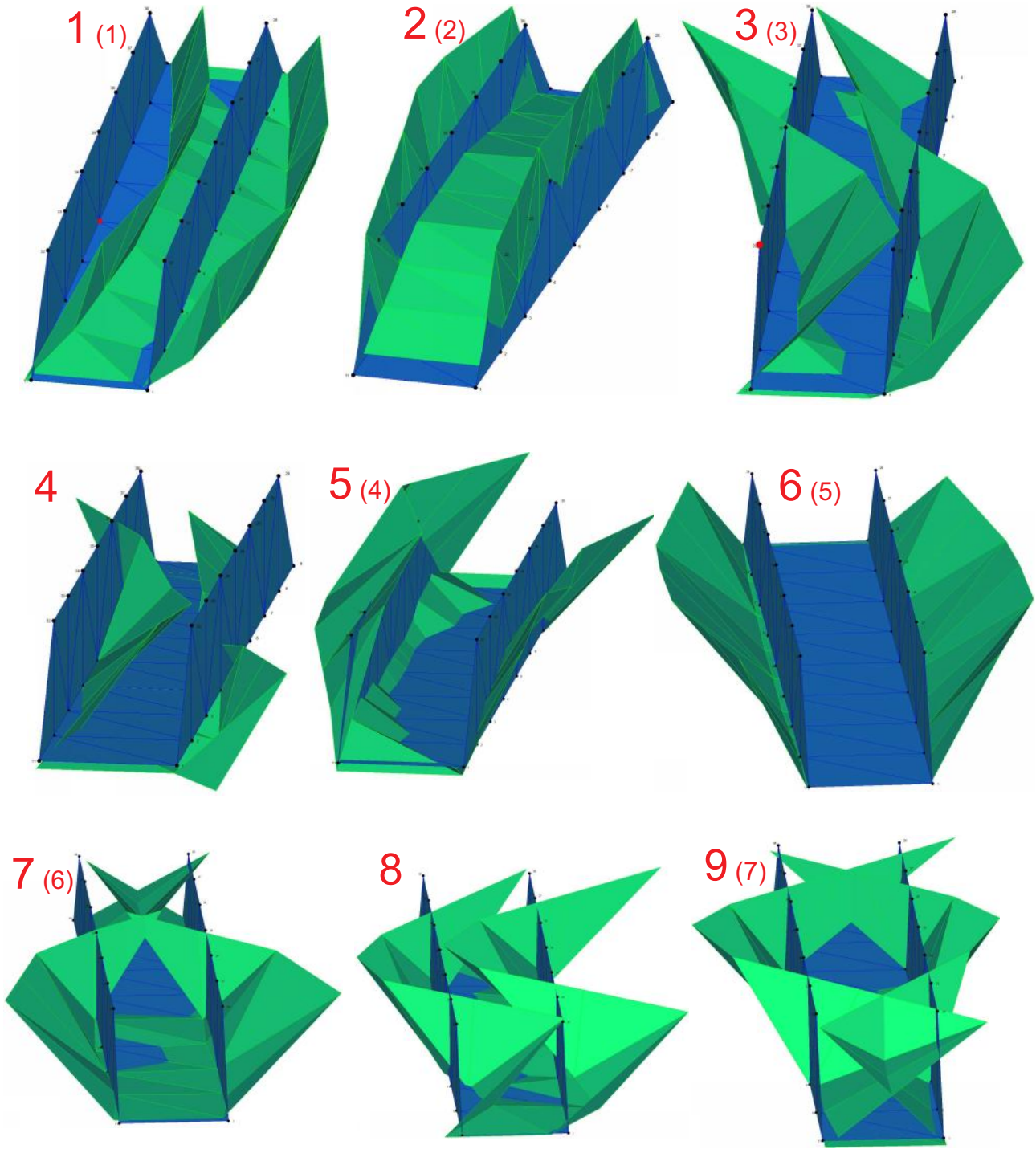
Figur 27: Bron utsattes avslutningsvis för gradvis ökad belastning upp till 1329 ton, då plastisk deformering inträdde. Övre balkarna vek sig inåt, flänsarna på de undre balkarna under domkrafterna vek sig och även betongfundamenten som bron vilade på hade fått sprickor.

Tabell 1: Jämförelse av modfrekvenser i Figur 22, 28 och 29.

Från FEM	Hel bro	Knäckt bro, CFDD	Knäckt bro, SSI-CVA
4,01	3,67	3,90	3,90
7,56	7,35	6,71	6,72
8,94	8,75	8,73	8,72
9,65	9,11	9,26	–
10,45	10,22	10,30	10,30
11,38	11,42	11,6	11,6
14,00	13,77	13,9	13,9
16,47	16,25	–	16,3
16,78	17,37	–	17,34

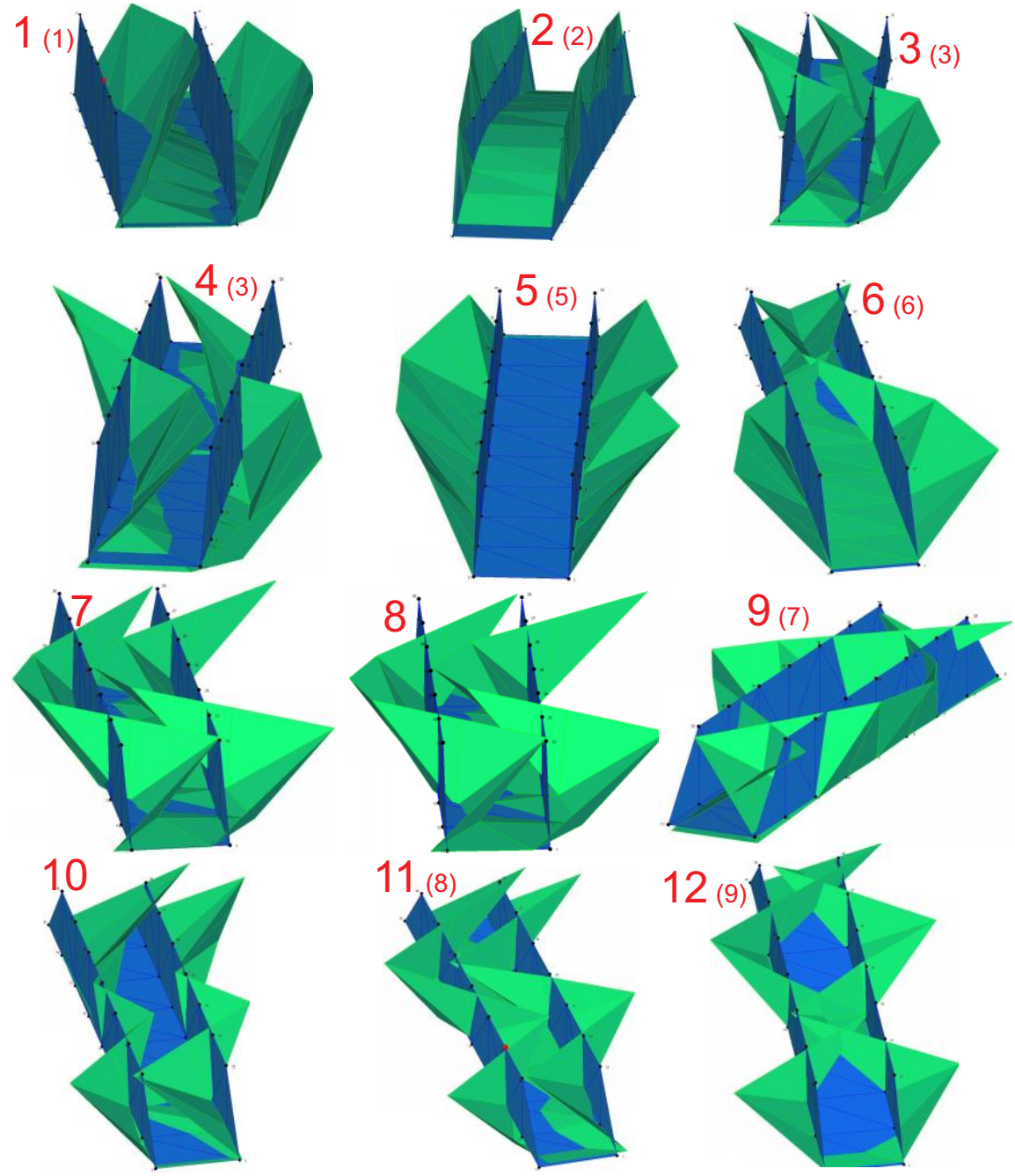
i modformer och de relativt små ändringarna i frekvenser räcker för att lokalisera skadorna, så kommer MATLAB-koden i Appendix D att modifieras för att kunna köras även på bron. Huvudalgoritmen för att lösa själva optimiseringsproblemet är oberoende av konstruktionens geometri, men andra modifieringar som krävs beskrivs i Avsnitt 3 och 4.1.

Mode	Frequency [Hz]	Std. Frequency [Hz]	Damping Ratio [%]	Std. Damping Ratio [%]	Comment	Creation Date & Time
CFDD Mode 1	3.898	0.02556	0.36	0.06975	Found Automatically	10-11-2013 01:54:52
CFDD Mode 2	6.713	0.04558	0.5256	0.1588	Found Automatically	10-11-2013 01:54:52
CFDD Mode 3	8.732	0.06455	0.2305	0.06409	Found Automatically	10-11-2013 01:54:52
CFDD Mode 4	8.954	0.07091	0.09873	0.05554	Found Automatically	10-11-2013 01:54:52
CFDD Mode 5	9.258	0.01468	0.1168	0.09546	Found Automatically	10-11-2013 01:54:52
CFDD Mode 6	10.3	0.01827	0.1941	0.09297	Found Automatically	10-11-2013 01:54:52
CFDD Mode 7	11.6	0.01042	0.08672	0.04438	Found Automatically	10-11-2013 01:54:52
CFDD Mode 8	12.74	0.08638	0.7925	0.2073	Found Automatically	10-11-2013 01:54:52
CFDD Mode 9	13.88	0.02577	0.09504	0.01583	Found Automatically	10-11-2013 01:54:52



Figur 28: Modformer och frekvenser motsvarade de i Figur 22 men efter att ha belastat bron till brott. Analyserat i ARTeMIS med metoden "Curve-fit Frequency Domain Decomposition". Inom parantes anges nummer för motsvarande modform i Figur 22.

Mode	Frequency [Hz]	Std. Frequency [Hz]	Damping Ratio [%]	Std. Damping Ratio [%]	Comment	Creation Date & Time
SSI-CVA Mode 1	3.895	0.02767	0.5528	0.7182	Found automatically	10-11-2013 00:12:56
SSI-CVA Mode 2	6.724	0.03804	0.9463	0.8092	Found automatically	10-11-2013 00:12:56
SSI-CVA Mode 3	8.724	0.0516	0.7423	0.4062	Found automatically	10-11-2013 00:12:56
SSI-CVA Mode 4	8.729	0.05225	0.7224	0.4155	Found automatically	10-11-2013 00:12:56
SSI-CVA Mode 5	10.3	0.01647	0.3064	0.2695	Found automatically	10-11-2013 00:12:56
SSI-CVA Mode 6	11.6	0.02049	0.2569	0.3276	Found automatically	10-11-2013 00:12:56
SSI-CVA Mode 7	12.74	0.1516	1.555	0.8122	Found automatically	10-11-2013 00:12:56
SSI-CVA Mode 8	12.76	0.1138	1.457	0.7543	Found automatically	10-11-2013 00:12:56
SSI-CVA Mode 9	13.88	0.02652	0.1575	0.116	Found automatically	10-11-2013 00:12:56
SSI-CVA Mode 10	15.11	1.886	1.435	0.646	Found automatically	10-11-2013 00:12:56
SSI-CVA Mode 11	16.28	0.05783	0.7464	0.3045	Found automatically	10-11-2013 00:12:56
SSI-CVA Mode 12	17.34	0.03738	0.2811	0.2166	Found automatically	10-11-2013 00:12:56



Figur 29: Samma som Figur 28, men analyserat i ARTeMIS med metoden "SSI Canonical Variate Analysis". Inom parantes anges nummer för motsvarande modform i Figur 22.

C Preliminära resultat för bron över Långforsen

På följande sidor bifogas en kopia av konferensbidraget [SGP+12], som redovisar en del preliminära resultat från de olika mätningarna och testerna på bron över Långforsen

The Railway Concrete Arch Bridge over Kalix River - Dynamic Properties and Load Carrying Capacity

Tekn. Lic. Natalia Sabourova,
Dr. Nikas Grip,
Tekn. Lic. Arto Puurula,
Tekn. Lic. Ola Enochsson,
Guest Prof. Dr. Yongming Tu
Ass. Prof., Dr Ulf Ohlsson,
Ass. Prof., Dr. Martin Nilsson,
Emer. Prof., Dr. Lennart Elfgren
Div. of Structural Engineering,
Luleå University of Technology,
SE-971 87 Luleå, Sweden
E-mail: firstname.secondname@ltu.se

Tekn. Dr. Anders Carolin,
Trafikverket,
Box 809,
SE- 971 25 Luleå, Sweden
E-mail: Anders.Carolin@ltu.se

ABSTRACT
The concrete bridge over Kalix river at Långforsen was built in 1960 and has a span of 90 m and a height of 13,7 m. The bridge owner, Trafikverket, now wants to increase its allowable axle load from 225 to 300 kN. Field tests were carried out under service condition and with ambient vibrations. The test results were used to update and validate Finite Element Models. At last, the refined models are used to check the possibility to increase the axle load.

Key words: Codes application, life-cycle assessment, modelling, methods and behaviour, monitoring, assessment and maintenance, numerical methods, sustainable structures.

1. INTRODUCTION

1.1 General

The Swedish Railways have a history of about 150 years and several of the bridges along the lines have been in service for more than 50 years, some even more than 100 years. The owner, Trafikverket, now wants to increase the maximum allowed axle load from 225 kN to 250 kN or more along the main lines in northern Sweden and this calls for an appraisal of the bridges.

Before higher axle load can be allowed it is necessary to check the maximum deflection and the general dynamic behaviour of the bridges. The concrete was designed to have a compressive strength of 40 MPa. Test on drilled out cores now show that the strength has increased to a mean strength of 56 MPa. In this paper, several finite element models of the Långforsen Bridge are discussed. Field tests are carried out under service condition and with ambient vibrations. The test results were used to update and validate finite element models. At last, the refined models are used to check the possibility to increase the axle load, Sabourova et al (2012).

1.2 Design and Construction

The bridge at Långforsen has a total length of 177.3 m with a central arch of 89.5 m and two side spans of 42 m, see Figure 1. It was built in 1960 and consists of an arch which carries a reinforced concrete slab via underlying longitudinal and transversal concrete beams, connected through fixed columns. The arch is a reinforced concrete box girder with two hollow spaces. The cross section is lowest at the crown and highest at the connection to the arch abutment. The original train load corresponds to an axle load of 250 kN for the locomotive and a distributed load of 72 kN/m.

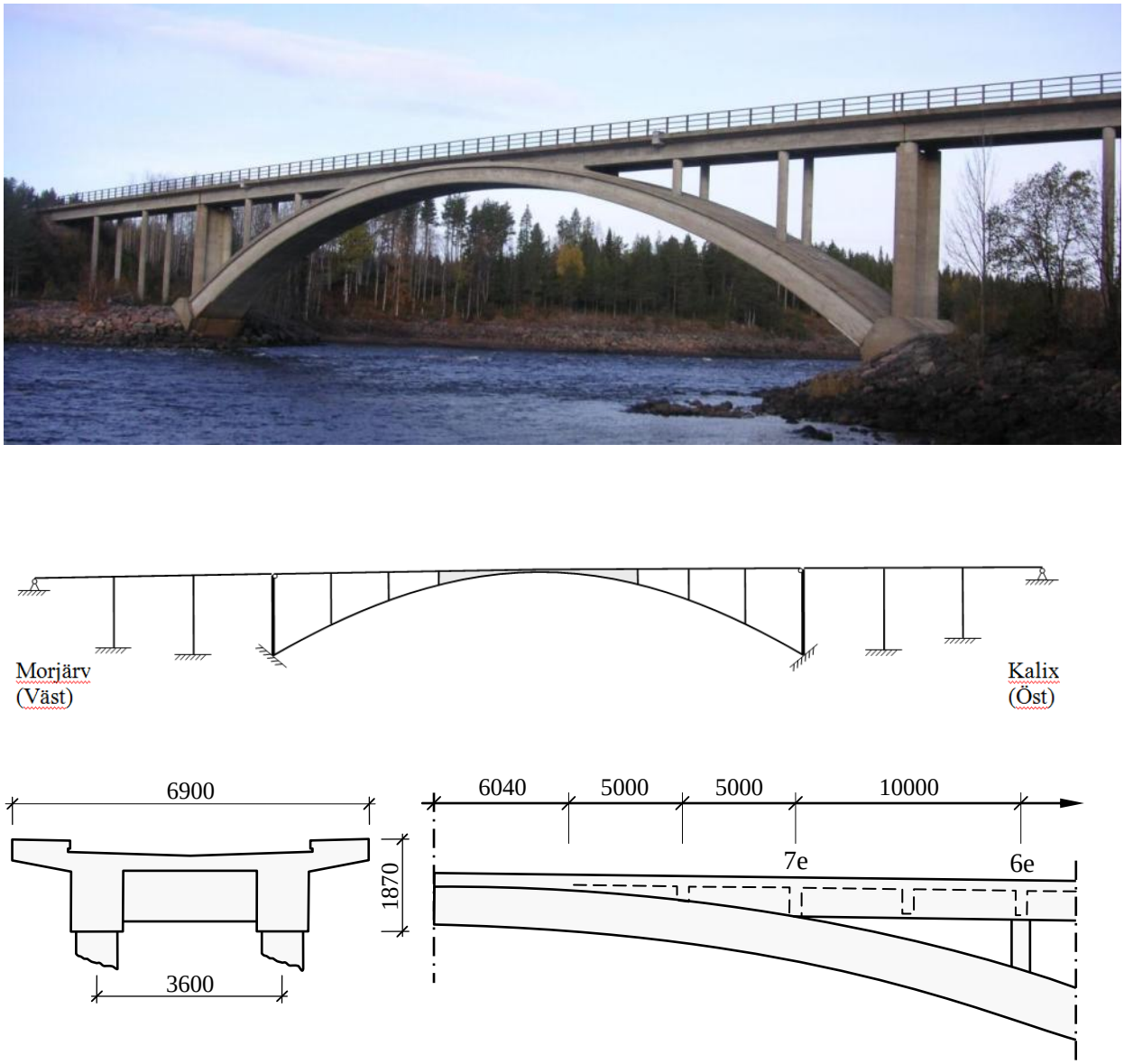


Figure 1. The Railway Concrete Arch Bridge over Kalix River at Långforsen.

2. GEOMETRY AND MATERIAL PROPERTIES

The bridge was built with concrete Btg K 400 with nominal characteristic compressive and tensile strengths of f_{cck} = 30.775 MPa and f_{ctk} = 1.95 MPa and with a modulus of elasticity of E_c = 32 GPa. The foundations were cast with a slightly lower concrete quality, Btg K 300. The reinforcement was ribbed bars Ks 40 and high strength steel Ss70 with yield strengths of f_{yk} = 390 and 720 MPa respectively.

In 2009 six cylinders with a diameter of 95 mm were drilled out of the lower part of the arch. Three were tested in compression giving compressive strengths of 76.7; 79. 8 and 65.2 MPa with a mean value of 73.9 MPa; three were splitted with splitting strengths of 1.85; 4.17 and 3.77 MPa. This corresponds to concrete quality K80 according to the Swedish Code BBK 94 with a characteristic compressive strength of f_{cck} = 56.5 MPa and a modulus of elasticity E_{ck} = 38.5 GPa.

3. FINITE ELEMENT MODELS

A lot of work has been invested in different FEM models over the years, Sabourova et al (2012). During 2011 two types of bridge models have been developed by Yongming Tu with Abaqus/Brigade: A comprehensive model with foundations (Type I) and a simplified model where the foundations have been exchanged by springs (Type II). The advantage of the former model is that it is closer to the real bridge structure, and the predicted results from it should be more reliable and closer to the 'real results', but the disadvantage is that the problem size is rather high. The number of elements is 93 910 in type I and decreases to 47 438 in type II and the number of variables decreases from 438 800 to 282 808. Both models give very close predictions of eigenfrequencies and deflections.

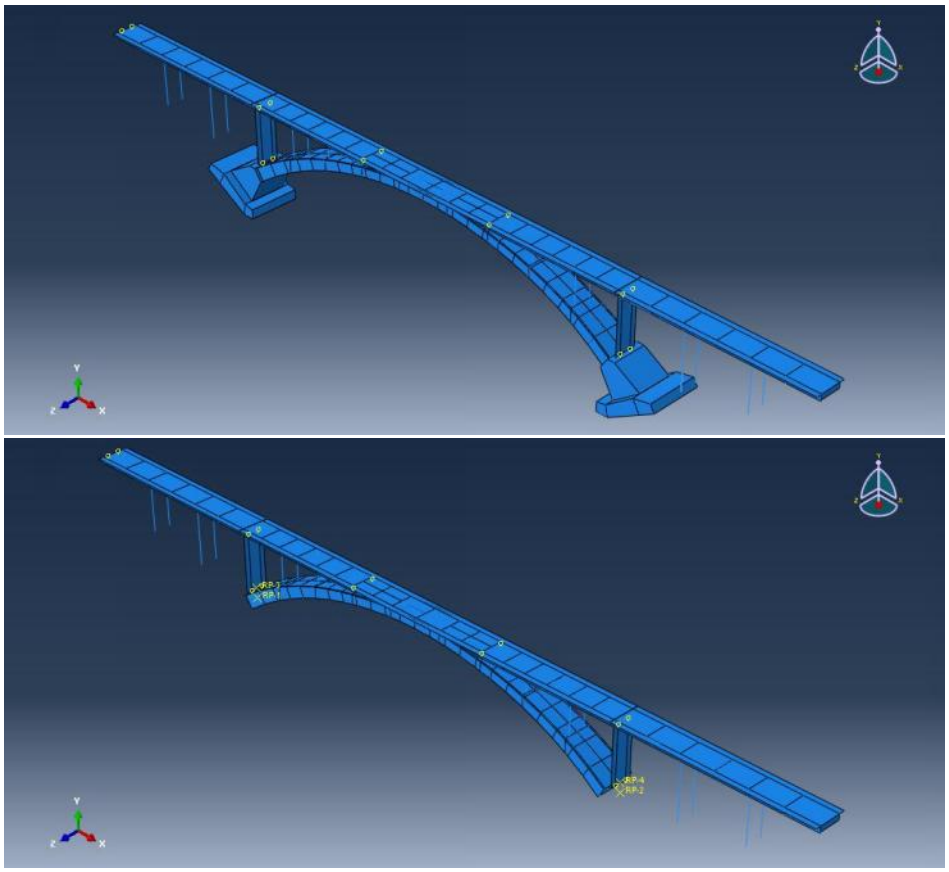


Figure 2. Two FEM models were used: A detailed model, where the foundations are included in the model (top); and a simplified model, where the foundations are simulated with springs (bottom).

4. MEASUREMENTS

Measurements of accelerations, strains and deflections have been carried out during 2009 and 2011. A simple non-iterative method to calibrate the accelerometers has been developed by Grip and Sabourova (2011) and further work is going on. A comparison between some measured and predicted eigenfrequencies with the Type II FEM Model are illustrated in Figure 3.

A comparison was made for some of the measured and predicted strains in a section in the top beams carrying the slab and the rail close to the centre of the arch. These sections had been critical in earlier assessments of the bridge. The maximum measured and predicted strains during the passing of a locomotive with an axle load of 180 kN were rather close and of the order of $\Delta\varepsilon = 20 \cdot 10^{-6}$ corresponding to a stress amplitude of only $\Delta\sigma = 4$ MPa.

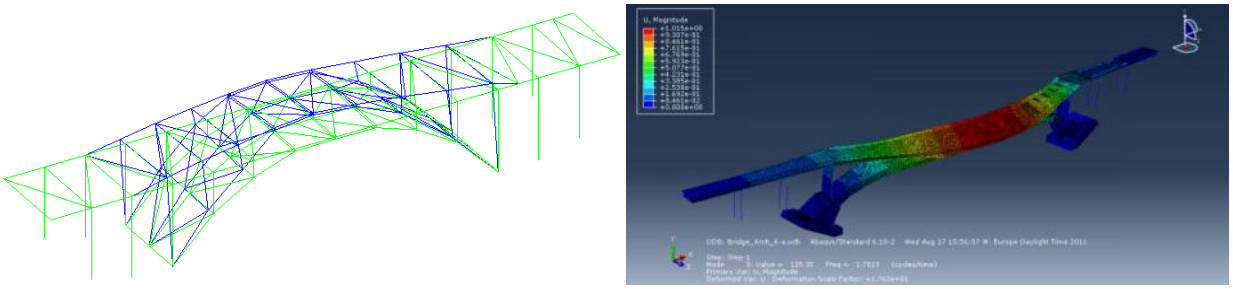


Figure 3. The first transverse bending eigenmode according to measurements gave a frequency 1.79 ±0.002 Hz (left). According to the FEM model we got the corresponding value 1.78 Hz (right). The difference is less than 1%.

5. LOAD CARRYING CAPACITY AND CONCLUSIONS

Methods for assessing the load carrying capacity have been developed in the EC-project Sustainable Bridges (2008), He et al (2008). These methods have been used here.
According to earlier assessments most parts of the bridge is capable of carrying an axle load of 330 kN. The only critical sections are located in the slab on top of the arch: (1) in the section where it is united with the arch, where the longitudinal top reinforcement is highly stressed; and (2) in the transverse direction between the supporting beams, where the bottom reinforcement is highly stressed. We are now studying these sections in the FEM model for different loads and preliminary results show low maximum stress ranges of 10 MPa in the critical sections and vertical deflections of the crown of the arch of the order of only ± 5 mm indicating that the bridge may be able to carry an increased axle load of 330 kN without problems.

ACKNOWLEDGEMENTS

The work has been supported by Formas, the Development Fund of the Swedish Construction Industry (SBUF), Trafikverket, Luleå Centre for Risk Analysis and Risk Management (CRR), the European Regional Funds and Luleå University of Technology.

REFERENCES

Grip, N., Sabourova, S.,2011
"Simple non-iterative calibration for triaxial accelerometers". Measurement Science and Technology, Vol 22, No 12, 12 pp.

He, G., Zou, Z., Enochsson, O., Bennitz, A., Elfgren, L., Kronborg, A., Töyrä, B., and Paulsson, B. (2008):
"Assessment of railway concrete arch bridge by numerical modelling and measurements". In "Bridge Maintenance, Safety, Management, Health Monitoring and Informatics" edited by H.-M. Koh and D. M. Frangopol. Leiden: CRC Press/Balkema, Taylor & Francis Group, ISBN 978-0-415-46844-2, Abstract p 722 + full version on CD pp 3733-3742.

Sabourova, N., Tu, Y., Grip, N., Enochsson, O., Ohlsson, U., Nilsson, M. C., and Elfgren, L. 2012.
"Railway Concrete Arch Bridge over Kalix River at Långforsen. Dynamic Properties and Load-Carrying Capacity". Research Report, Div. of Structural Engineering, Luleå University of Technology, Luleå. To be published.

Sustainable Bridges, 2008
"Sustainable Bridges – Assessment for Future Traffic Demands and Longer Lives". A European Integrated Research Project during 2003-2008. Four guidelines and 35 background documents are available at www.sustainablebridges.net

D Tillståndsbedömning via FEM-uppdatering i Matlab

Vi redovisar här en förenklad version av MATLAB-koden för FEM-uppdatering som användes för att ta fram resultaten i Avsnitt 2.2. Vidare förenklingar av programstrukturen pågår när detta skrivs för senare publicering. Sedan kommer framför allt rutiner för indata, utdata, plottning av resultat samt `InterpolFunctions`-delen att generaliseras för andra konstruktioner, först bron över Åby älv.

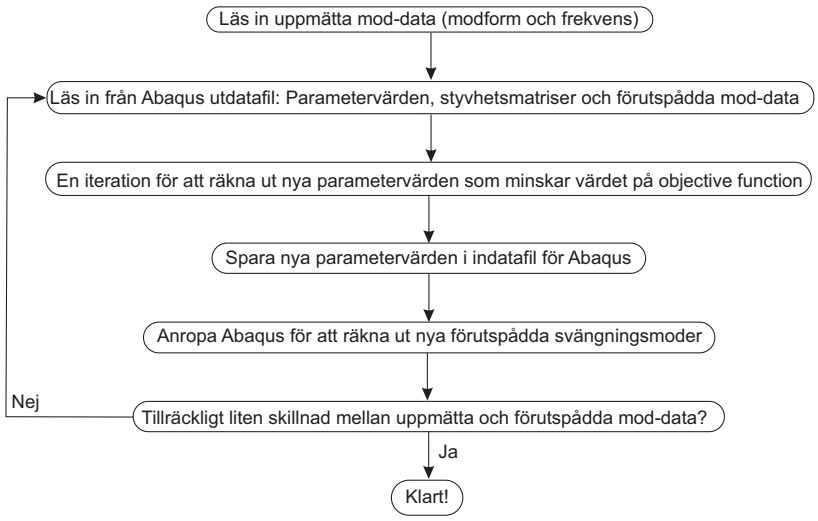
Själva huvudalgoritmen är dock oberoende av konstruktionens geometri och därmed samma även för andra konstruktioner. Den använder bland annat optimiseringsmetoden "Newton trust region" för att justera nyckelparametrar i en FEM-modell av konstruktionen för att hitta ett lokalt minimum för "avståndet" mellan uppmätta och av FE-modellen förutspådda modformer och -frekvenser.

De implementerade analysteknikerna finns beskrivna i olika forskningsartiklar och böcker som [FM95], men det finns ingen fritt tillgänglig mjukvaruimplementering eller utförlig pseudokod som gör metoderna lätttillgängliga för de som vill tillämpa FEM-uppdatering på egna mätdata och finita elementmodeller. Detta är ett första steg i den riktningen.

Slutversionen av koden som presenteras här är tänkt att dels göra FEM-uppdatering mer lättillgänglig och dels stödja reproducerbar forskning samt underlätta utveckling av nya analysmetoder.

Vi ger ingen närmare beskrivning här av de delar av koden som läser utdatafiler från ARTeMIS och Abaqus aamt skriver nya indatafiler för Abaqus. Fokus är istället på att visa hur vi implementerat ett iterationssteg i den del av koden (se Figur 30) som löser själva optimiseringsproblemet. Källkod skriven i till exempel Fortran or C/C++ hade gett snabbare exekvering, men vi har valt att skriva i MATLAB för att det underlättar att snabbt sätta sig in i koden och modifiera efter egna behov. Även om MATLAB är komersiellt så finns gratis "MATLAB klioner" som Octave and Scilab för de som vill återanvända kod utan att behava betala för MATLAB. Koden kräver dock att man har FEM-programmet Abaqus om man vill köra den på den modell för betongplattan som används här.

Vi beskriver de analyssteg som utförs till viss del och hänvisar till exempel till boken [NW06, Kapitel 4] för en relativt lättbegriplig av den optimiseringsmetod ("Newton trust region met-



Figur 30: (Kopia av Figur 3.) Optimisering med metoden "Newton trust region". Skillnaden mellan uppmätta och förutspådda mod-datat mäts av den målfunktion (objective function) som definieras i (7) nedan.

hod") som vi använder och [FM95, Mar10] för en översikt över olika FEM-uppdateringstekniker.

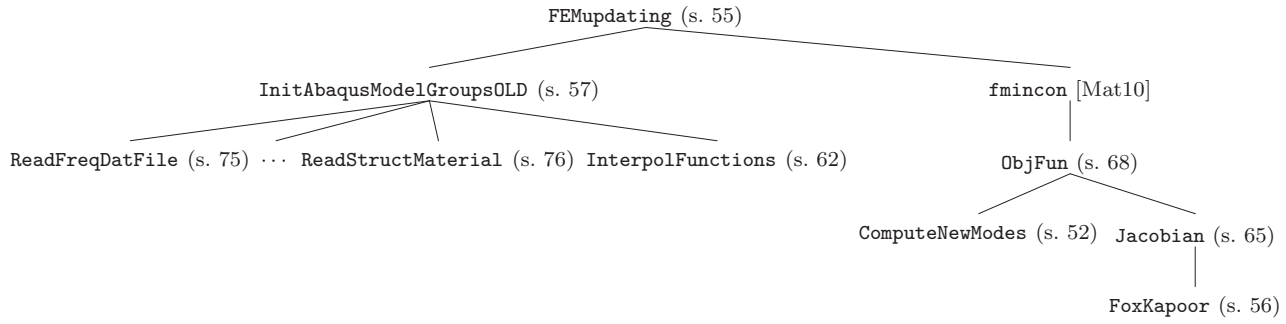
Slutmålet är att ha kod som enkelt kan modifieras och expanderas för olika konstruktioner och andra optimiseringsproblem. I sin nuvarande form är den skriven för att fungera på finita elementmodeller under följande villkor:

- Uppdaterade parametrar (som elasticitetsmodul) är begränsade till att ligga i ett intervall som bestäms innan programmet startas.
- Det startar från ett givet eller slumpartat valt startvärde på uppdateringsparametrar och söker sig fram till ett lokalt minimum. (För global optimisering så finns metoder som "Coupled Local Minimizers" som startar parallellt från flera olika startvärden på parametrarna.)
- Vi uppdaterar materialegenskaper, inte geometri (vilket är svårare).
- Varje nodpunkt i finita elementmodellen har samma antal frihetsgrader.
- En valbar möjliget är regularisering av resultaten med så kallade "damage functions", som dock är begränsade till rektangulärt utplacerade nodpunkter.

  Grundidén är då att "släta ut" det resulterande valet av uppdateringsparametrar genom att först låta FEM-uppdateringsalgoritmen välja parametervärden fritt för en delmängd av de grupper av element som har uppdateringsbar parameter (inklusive hörnpunkter), och sedan låter parametervärden för övriga grupper bestämmas med någon form av inpterpolation. En enkel teknik för detta är FEM-uppdatering via så kallade "shape functions" föreslogs i [TMDR02] för till exempel balkar. Artikeln [SD+09] ger några förslag till hur dessa kan generaliseras till "shape functions" definierade på en yta. Vi har gjort det senare i funktionen `InterpolFunctions`, vilket fungerar bra för betongplattan i Avsnitt 2.2, men för mer komplicerade geometrier krävs generaliseringar.

D.1 Implementering i Matlab

Den huvudsakliga programstrukturen i MATLAB-implementationen visas i detta träddiagram:



Huvudfunktionen anropas med indataparametrar som man enklast får lämpliga värden på från funktionen `InputParameters`, med följande kommandon i MATLAB:

```
>InputParameters
>FemUpdating(AbaqusInpFilName,MasterFemNodes,MasterFemDofs,MasterMeasDofs,...
             nuMeas,phiMeas,GroupNames,Pmin,Pmax,P0,CoarseMeshInd,MeasX,...
             MeasY,nX,nY,MeasDirection,NrOfDofsPerNode,RefNode,RefDof,...
             SparseMtx,ShapeResType,WeightingStrategyModePairingMethod,...
             ModeScalingMethod,W,PlotModeShapes)
```

De 16 första indatavärdena är nödvändiga. Återstående sätts automatskt till olika defaultvärden om de ej anges. Funktionen `FEMupdating` anropar först funktionen `InitAbaqusModelGroupsOLD` som returnerar något tiotal konstanta parametervärden i en struktur `OptConst` samt sätter startvärden på de parametrar som uppdateras i varje iteration och returnerar dessa i en struktur `StructUpdating` för somliga startvärden/konstanter. För den senare annropas bland annat `ReadStructMaterial`, `BuildGlobalElemNr2GrNrLokup_NEW2`, `InterpolFunctions`, `ReadElementStiffMtxFile` samt kommandot `UpdateAbaqusInpFile` för att de valda startvärden och konstanter som påverkar FEM-modellen sätts till aktuella värden även där.

Vi går ej närmare genom de funktionerna som anropas av `InitAbaqusModelGroupsOLD` i detta appendix, utan beskriver främst FEM-uppdateringsalgoritmen som följer sedan. För detta anropar `FEMupdating` Matlab Optimization Toolbox-funktionen `fmincon`, som söker sig fram ett värde på parametrarna i `StructUpdating` som ger ett lokalt minimum för "avståndet" mellan uppmätta och förutspådda mod-data (modformer och -frekvenser). Avståndet mäts med en så kallad "objective function" som beskrivs och definieras i (7) nedan. En introduktion till hur "Newton trust region"-algoritmen fungerar finns i [NW06], men i korthet så söker den sig till ett lokalt minimum för målfunktionen genom att i varje iterationssteg approximera denna med ett Taylorpolynom av grad 2 och använda detta för att räkna ut lämplig justering i rätt riktning av parametrar i `StructUpdating`. (Som nämnts i avsnitt 4.1.1 så finns alternativa optimiseringsalgoritmer, som den i Appendix D.3.24, som ej untyttjar andraderivator, vilket kan vara ett alternativ om man har svårt få bra uppskattningar av andraerivator av målfunktionen).

För Taylorapproximation av grad två behöver `fmincon` aktuellt värde, derivator och andraderivator av målfunktionen som fås genom att anropa funktionen `ObjFun`. `ObjFun` anropar först `ComputeNewModes`, som i sin tur anropar Abaqus, som räknar ut nya förutspådda mod-data utifrån aktuella parametervärden i `StructUpdating`. Sedan fås uppskattningar av derivator och andraderivator genom att anropa funktionen `Jacobian`, som i sin tur beföver hjälpfunktionen `FoxKapoor` för uppskattning av förstaderivatorna. Hur dessa uppskattningar görs beskrivs närmare i avsnitt D.2. Avslutningsvis anropar `ObjFun` funktionen `ComputeNewModes`, som i sin tur anropar Abaqus, som räknar ut nya förutspådda mod-data utifrån de justeringar som gjorts av parametrar i `StructUpdating`.

När detta är gjort får `fmincon` tillbaka kontrollen, kan justera parametrarna i `StructUpdating` som beskrivs i [NW06], och sedan gå vidare till nästa iteration i "Newton trust region"-optimeringen, alternativt avbryta om värdet på målfunktionen blivit tillräckligt litet (se Figur 30).

Vi ger närmare beskrivning av nödvändiga derivatauppskattningar i Appendix D.2. Sedan beskriver vi kortfattat de de funktiner som implementerar dessa uppskattningar i MATLAB.

D.2 Parametrar och derivataberäkning

Vi sammanfattar använd notation i Tabell 2. Konstruktionens finita elementmodell delas in i G grupper om (ett eller flera) finita element som alla har samma värde på den uppdaterade parametern (till exempel elasticitetsmoidul). Vi vill uppdatera startvärden $X_{0,g}$ på på den uppdaterade fysikaliska parametern till nytt värde X_g. Vi uppdaterar med en multiplikativ faktor $1 - a_g$, så att

$$X_e = X_{0,g}(1 - a_g), \qquad g = 1, \ldots, G.$$

Metoden är begränsad till att uppdatera fysikaliska parametrar, som elasticitetsmodul, sådana att grupp-styvhetsmatrisen K_g beror som en linjär funktion av X_e. Alltså uppdateras styvhetsmatrisen på motsvarande sätt:

$$K_g = K_{0,g}(1 - a_g).$$

Den globala styvhetsmatrisen är då

$$K = K_u + \sum_{g=1}^{G} K_{0,g}(1 - a_g), \tag{4}$$

där K_u är summan av alla oförändrade grupp-styvhetsmatriser.

Vi kallar K den (globala) styvhetsmatrisen och K_g grupp-styvhetsmatrisen, eller den utvidgade grupp-styvhetsmatrisen om vi vill klargöra att den är utfulld med nollor till samma storlek som K.

För uppdateringsparametrarna, sparsde i en vektor $\boldsymbol{a}$, så är målet att välja $\boldsymbol{a}$ som minimerar vissa frekvens- och modformsresidualer $\boldsymbol{r}_\mathbf{f}(\boldsymbol{a})$ och $\boldsymbol{r}_\mathbf{s}(\boldsymbol{a})$. Frekvensresidualen är en vektor med element

$$(\boldsymbol{r}_\mathbf{f}(\boldsymbol{a}))_m \stackrel{\text{def}}{=} \frac{(2\pi\nu_m^{\text{FEM}}(\boldsymbol{a}))^2 - (2\pi\nu_m^{\text{mea}})^2}{(2\pi\nu_m^{\text{mea}})^2} = \frac{(\nu_m^{\text{FEM}}(\boldsymbol{a}))^2 - \nu_m^{\text{mea}2}}{\nu_m^{\text{mea}2}}, \qquad m = 1, \ldots, M. \tag{5}$$

Modformsresidualen jämför uppmätta och förutspådda modformer i samma nodpunkter i konstruktionen/modellen, men det är viktigt att jämföra *formerna*, oberoende av amplitud. Både uppmätta och förutspådda modformer normaliseras därför så att

$$(\boldsymbol{r}_\mathbf{s}(\boldsymbol{a}))_{m,n} = \frac{\phi_m^{\text{FEM}}(\boldsymbol{a})}{\|\boldsymbol{\phi}_m^{\text{FEM}}(\boldsymbol{a})\|_2} - \frac{\phi_{m,n}^{\text{mea}}}{\|\boldsymbol{\phi}_m^{\text{mea}}\|_2}, \qquad m = 1, \ldots, M \qquad n = 1, \ldots, N. \tag{6}$$

Den normeringen avviker från varianter vi har sett tidigare och har fördelen att varje modform ger ett bidrag mindre än två till målfunktionen (7), och alltså viktas ungefär lika högt som varje modfrekvensfel i $\boldsymbol{r}_\mathbf{f}$. Detta bör vara en bra grund för eventuella ytterligare justeringar med vikterna i matrisen W i (10) nedan. (Ett alternativ till denna modformsresidual $\frac{\boldsymbol{\phi}_{d,m}^{\text{FEM}}(\boldsymbol{a})^{\mathrm{T}}\boldsymbol{\phi}_{d,m}^{\text{mea}}}{\|\boldsymbol{\phi}_{d,m}^{\text{FEM}}(\boldsymbol{a})\|_2^2}$, som användes under namnet "*modal scale factor* i [RTDR10]. Några andra vanliga mått för att jämföra modformer presenteras i [Jai05, Section 3.2].

För $R = M + MDN = M(DN + 1)$ och $\boldsymbol{r}(\boldsymbol{a}) = \begin{pmatrix} r_1(\boldsymbol{a}) & \cdots & r_R(\boldsymbol{a}) \end{pmatrix}^{\mathrm{T}} = \begin{pmatrix} \boldsymbol{r}_\mathbf{f}(\boldsymbol{a}) \\ \boldsymbol{r}_\mathbf{s}(\boldsymbol{a}) \end{pmatrix}$, är vårt mål att välja ett $\boldsymbol{a}$ som minimerar *målfunktionen* (*objective function*)

$$f(\boldsymbol{a}) = \frac{1}{2}\boldsymbol{r}(\boldsymbol{a})^{\mathrm{T}} W \boldsymbol{r}(\boldsymbol{a}) = \frac{1}{2}\left(w_1 r_1(\boldsymbol{a})^2 + \cdots + w_R r_R(\boldsymbol{a})^2\right), \tag{7}$$

Tabell 2: Notation för vektorer och indexmängder som används i detta appendix och i MATLAB-implementeringen. Totala antalet frihetsgrader N är lika med antalet accelerometer-mätpunkter multiplicerade med antalet frihetsgrader för varje nodpunkt i finita elementmodellen. Nuvarande implementation är begrensad till finita elementmodeller där varje nodpunkt har samma antal frihetsgrader.

$\boldsymbol{a} = \begin{pmatrix} a_1 & \cdots & a_G \end{pmatrix}^{\mathrm{T}}$	Decision parameter vector
$g = 1, \ldots, G$	G är antalet $\boldsymbol{g}$upper av element.
$m = 1, \ldots, M$	M är antalet $\boldsymbol{m}$oder som skall jämföras.
$n = 1, \ldots, N$	N är totala antalet frihetsgrader i varje vibrationsmod.
$\boldsymbol{\phi}^{\text{mea}}{}_m = \begin{pmatrix} \phi_{m,1}^{\text{mea}} & \cdots & \phi_{m,N}^{\text{mea}} \end{pmatrix}$	Den $m$te uppmätta modformen.
$\boldsymbol{\phi}_m^{\text{FEM}}(\boldsymbol{a}) = \begin{pmatrix} \phi_{m,1}^{\text{FEM}}(\boldsymbol{a}) & \cdots & \phi_{m,N}^{\text{FEM}}(\boldsymbol{a}) \end{pmatrix}^{\mathrm{T}}$	Den $m$te förutspådda modformen.
ν_m^{mea}	Den $m$tte uppmätta modfrekvensen.
$\nu_m^{\text{FEM}}(\boldsymbol{a})$	Den $m$te förutspådda modfrekvensen.

där W är en diagonal matris med diagonalelement w_ρ

För minimering av f med metoden Newton trust region Trust region så behövs Taylor polynom-approxiamtionen

$$f(\boldsymbol{a_0}+\boldsymbol{h}) \approx f(\boldsymbol{a_0}) + (\nabla f)(\boldsymbol{a_0})^{\mathrm{T}}\boldsymbol{h} + \frac{1}{2}\boldsymbol{h}^T H_f(\boldsymbol{a_0})\boldsymbol{h} \stackrel{\text{def}}{=} m(\boldsymbol{a}). \tag{8}$$

Från denna och (7) kan gradienten ∇f och Hessianen H_f beräknas som följer. Gradienten är $E \times 1$-vektorn där element nummer e är

$$\begin{aligned}(\nabla f)_e =& \frac{\partial f}{\partial a_g} = w_1 r_1(\boldsymbol{a})\frac{\partial r_1}{\partial a_g} + \cdots + w_R r_R(\boldsymbol{a})\frac{\partial r_R}{\partial a_g} \\ \nabla f =& \sum_{\rho=1}^{R} w_\rho r_\rho(\boldsymbol{a})(\nabla r_\rho)(\boldsymbol{a}) = J_{\boldsymbol{r}}^{\mathrm{T}}(\boldsymbol{a}) W \boldsymbol{r}(\boldsymbol{a}), \qquad \text{for the } \textit{Jacobian } (J_{\boldsymbol{r}}(\boldsymbol{a}))_{\rho,g} \stackrel{\text{def}}{=} \frac{\partial r_\rho}{\partial a_g}\end{aligned} \tag{9}$$

På liknande sätt fås Hessianen

$$\begin{aligned}(H_f)_{k,l} \stackrel{\text{def}}{=} \frac{\partial^2 f}{\partial a_k \partial a_l} &= \frac{\partial^2}{\partial a_k \partial a_l}\frac{1}{2}\left(w_1 r_1(\boldsymbol{a})^2 + \cdots + w_R r_R(\boldsymbol{a})^2\right) = \frac{\partial}{\partial a_k}\sum_{\rho=1}^{R} w_\rho r_\rho(\boldsymbol{a})\frac{\partial r_\rho(\boldsymbol{a})}{\partial a_l} \\ &= \sum_{\rho=1}^{R}\left(w_\rho \frac{\partial r_\rho(\boldsymbol{a})}{\partial a_k}\frac{\partial r_\rho(\boldsymbol{a})}{\partial a_l} + w_\rho r_\rho(\boldsymbol{a})\frac{\partial^2 r_\rho(\boldsymbol{a})}{\partial a_k \partial a_l}\right) = \sum_{\rho=1}^{R}\left(w_\rho \frac{\partial r_\rho(\boldsymbol{a})}{\partial a_k}\frac{\partial r_\rho(\boldsymbol{a})}{\partial a_l} + w_\rho r_\rho(\boldsymbol{a})(H_{r_\rho}(\boldsymbol{a}))_{k,l}\right) \\ H_f =& J_{\boldsymbol{r}}(\boldsymbol{a})^{\mathrm{T}} W J_{\boldsymbol{r}}(\boldsymbol{a}) + \sum_{\rho=1}^{R} w_\rho r_\rho(\boldsymbol{a}) H_{r_\rho}(\boldsymbol{a}) \approx J_{\boldsymbol{r}}(\boldsymbol{a})^{\mathrm{T}} W J_{\boldsymbol{r}}(\boldsymbol{a})\end{aligned} \tag{10}$$

Vektorn $\boldsymbol{r_f}$ har längd M, medan vektorn $\boldsymbol{r_s}$ har längd DMN med någon omordning $\rho(m,n)$ av index (m,n) för att spara elementen i $\boldsymbol{r_s}$ ien väldefinierad ordning i MATLAB-implementationen. Men exakt hur de ordnas har ingen betydelse i följande beräkningar. (Vilken inverterbar avbildning som helst av $\{1,2,\ldots,M\}\times\{1,2,\ldots,N\}$ på $\{1,2,\ldots,MN\}$ duger. För $M(N+1)\times E$-Jakobianen $J_{\boldsymbol{r}} = \begin{bmatrix} J_{\boldsymbol{r_f}} \\ J_{\boldsymbol{r_s}} \end{bmatrix}$, kommer vi att beteckna elementen $(J_{\boldsymbol{r_f}})_{m,g} \stackrel{\text{def}}{=} (J_{\boldsymbol{r}})_{m,g}$ och $(J_{\boldsymbol{r_s}})_{m,n;g} \stackrel{\text{def}}{=} (J_{\boldsymbol{r}})_{M+\rho(m,n),g}$. Från (5) får vi

$$(J_{\boldsymbol{r_f}})_{m,g} \stackrel{\text{def}}{=} \frac{\partial(\boldsymbol{r_f})_m}{\partial a_g} = \frac{1}{(2\pi\nu_m^{\text{mea}})^2}\frac{\partial(2\pi\nu_m^{\text{FEM}})^2}{\partial a_g}, \qquad m = 1,\ldots,M.$$

Det följer att $\frac{\partial\left\|\boldsymbol{\phi}_{d,m}^{\text{FEM}}\right\|_2}{\partial a_e} = \frac{\partial\left(\boldsymbol{\phi}_{d,m}^{\text{FEM}\,\mathrm{T}}\boldsymbol{\phi}_{d,m}^{\text{FEM}}\right)^{1/2}}{\partial a_e} = \frac{1}{2}\frac{2\frac{\partial\boldsymbol{\phi}_{d,m}^{\text{FEM}\,\mathrm{T}}}{\partial a_e}\boldsymbol{\phi}_{d,m}^{\text{FEM}}}{\left\|\boldsymbol{\phi}_{d,m}^{\text{FEM}}\right\|_2}$ och

$$\begin{aligned}(J_{\boldsymbol{r_s}})_{d,m,n;e} \stackrel{\text{def}}{=} \frac{\partial(\boldsymbol{r_s})_{d,m,n}}{\partial a_e} &= \frac{\frac{\partial\phi_{d,m,n}^{\text{FEM}}}{\partial a_e}\left\|\boldsymbol{\phi}_{d,m}^{\text{FEM}}\right\|_2 - \phi_{d,m,n}^{\text{FEM}}\frac{\partial\left\|\boldsymbol{\phi}_{d,m}^{\text{FEM}}\right\|_2}{\partial a_e}}{\left\|\boldsymbol{\phi}_{d,m}^{\text{FEM}}\right\|_2^2} \\ &= \frac{\frac{\partial\phi_{d,m,n}^{\text{FEM}}}{\partial a_e}\left\|\boldsymbol{\phi}_{d,m}^{\text{FEM}}\right\|_2 - \phi_{d,m,n}^{\text{FEM}}\frac{\frac{\partial\boldsymbol{\phi}_{d,m}^{\text{FEM}\,\mathrm{T}}}{\partial a_e}\boldsymbol{\phi}_{d,m}^{\text{FEM}}}{\left\|\boldsymbol{\phi}_{d,m}^{\text{FEM}}\right\|_2}}{\left\|\boldsymbol{\phi}_{d,m}^{\text{FEM}}\right\|_2^2} = \frac{\frac{\partial\phi_{d,m,n}^{\text{FEM}}}{\partial a_e}}{\left\|\boldsymbol{\phi}_{d,m}^{\text{FEM}}\right\|_2} - \frac{\frac{\partial\boldsymbol{\phi}_{d,m}^{\text{FEM}\,\mathrm{T}}}{\partial a_e}\boldsymbol{\phi}_{d,m}^{\text{FEM}}\phi_{d,m,n}^{\text{FEM}}}{\left\|\boldsymbol{\phi}_{d,m}^{\text{FEM}}\right\|_2^3}.\end{aligned}$$

De här uttrycken beror av derivatorna $\frac{\partial(2\pi\nu_m^{\text{FEM}})^2}{\partial a_g}$ och $\frac{\partial\boldsymbol{\phi}_m^{\text{FEM}}}{\partial a_g}$ som vi kan räkna ut, som i [TMDR02], från (4) med formler först härledda av Fox–Kapoor [FK68, eqs. (12),(18),(20)], för vårt speci-

alfall där enbart styvhetsmatrisen påverkas av parametrarna a_g:

$$\frac{\partial(2\pi\nu_m^{\mathrm{FEM}})^2}{\partial a_g} = \boldsymbol{\phi}_m^{\mathrm{FEM}^{\mathrm{T}}} \frac{\partial K}{\partial a_g} \boldsymbol{\phi}_m^{\mathrm{FEM}} = -\boldsymbol{\phi}_m^{\mathrm{FEM}^{\mathrm{T}}} K_{0,g} \boldsymbol{\phi}_m^{\mathrm{FEM}} \tag{11a}$$

$$\frac{\partial \boldsymbol{\phi}_m^{\mathrm{FEM}}}{\partial a_g} = \sum_{q \neq m} \frac{\boldsymbol{\phi}_q^{\mathrm{FEM}^{\mathrm{T}}} \frac{\partial K}{\partial a_g} \boldsymbol{\phi}_m^{\mathrm{FEM}}}{(2\pi\nu_m^{\mathrm{FEM}})^2 - (2\pi\nu_q^{\mathrm{FEM}})^2} \boldsymbol{\phi}_q^{\mathrm{FEM}} = \sum_{q \neq m} \frac{\boldsymbol{\phi}_q^{\mathrm{FEM}^{\mathrm{T}}} K_{0,g} \boldsymbol{\phi}_m^{\mathrm{FEM}}}{(2\pi\nu_q^{\mathrm{FEM}})^2 - (2\pi\nu_m^{\mathrm{FEM}})^2} \boldsymbol{\phi}_q^{\mathrm{FEM}} \tag{11b}$$

Remark 1. För stora E och för a_g långsamt varierande med g, så kan Jakobianen $J_{\boldsymbol{r}}$ vara illa konditionerad. För att undvika detta har det föreslagits i [RTDR10, TMDR02] att approximera a_g med summan $a_g \approx \sum_k c_k b_k(g)$ för ett mindre antal basfunktioner $b_k(g)$. Till exempel kan $b_k(g)$ vara så kallade tältfunktioner som ger styckvis linjära a_g.

D.3 Matlab-kod

Här redovisas funktionerna i Avsnitt D.1 i bokstavsordning med kortfattad beskrivning av vad de gör.

D.3.1 BuildGlobalElemNr2GrNrLokup_NEW2.m

Anropas av `InitAbaqusModelGroupsOLD.m` för att läsa in en del konstanter och parametervärden från utdatafil från Abaqus.

```
function [NoOfModes,NoOfNodes,GlobalElem2GrNrMap,GroupCenterPts] = ...
                    BuildGlobalElemNr2GrNrLokup_NEW2(AbaqusDatFile,...
                                                     AbaqusInputFile,...
                                                     GrNr2GrNameMap)
% INPUT:
% AbaqusDatFile   - Abaqus ".dat" output file with mapping between local
%                   and global element numbers
% AbaqusInputFile - Abaqus ".inp" input file with geometry and material
%                   properties definitions
% GrNr2GrNameMap  - group number to group name mapping

% OUTPUT:
% NoOfNodes  - total number of nodes in the Abaqus model
% NoOfGroups - number of groups
% GlobalElem2GrNrMap - table indexed by global element numbers and with
%                      entries saying to which group each particular element
%                      belongs
% GroupYoungModulus - Young modulus for the groups to be updated

% USAGE:
% Example:
% >>AbaqusDatFile = 'Bridge_Arch_6-a-sim2-Mea-FirstIterWithGlobalMassStiff.dat';
% >>AbaqusInputFile = 'Bridge_Arch_6-a-sim2-Mea-FirstIterWithGlobalMassStiff.inp';
% >>GrNr2GrNameMap = {'Concrete'};
% >>[NoOfNodes NoOfGroups GlobalElem2GrNrMap GroupYoungModulus] = ...
%                         BuildGlobalElemNr2GrNrLokup(...,
%                                                     AbaqusDatFile,...
%                                                     AbaqusInputFile,...
%                                                     GrNr2GrNameMap);


NoOfNodes = 0;
NameDelimiter = '*';


%% Read Abaqus ".dat" file and build a mapping between the local and
% the global element numbers
Local2GlobalPattern = '(LOCAL)\s+TO\s+GLOBAL\s+NODE\s+AND\s+ELEMENT\s+MAPS';
NodeNodePattern = '\s*(node)\s+node\s*';
```

```
ElemElemPattern = '\s*(element)\s+element\s*';
%InstanceNamePattern = '\s+(Instance name)';
InstNameNumNumPattern = '(?<name>[a-zA-Z0-9-_]+)\s+(?<num1>\d+)\s+(?<num2>\d+)\s*';

Tokens1 = cell(1);
Tokens1{1} = 'LOCAL';
Tokens1{2} = 'node';
Tokens1{3} = 'element';
%Tokens1{4} = 'Instance name';

InstLocal2GlobalNodeNrsMap = cell(1,2);
InstLocal2GlobalElemNrsMap = cell(1,2); % Instance name, global element numbers
                                % We use the fact that in each instance
                                % local element numbers start with 1 and
                                % are written in the ".dat" file in the
                                % sequential order.
                                % InstLocal2GlobalElemNrsMap is probably a
                                % better name

fid = fopen(AbaqusDatFile);
tline = fgetl(fid);
pattern = Local2GlobalPattern;
while ischar(tline) % if tline is array of characters

  [mat tok] = regexp(tline,pattern,'match','tokens');
  if ~isempty(tok)
    index = strmatch(tok{1}{1},Tokens1,'exact');
    switch index
      case 1 % Reading local to global node and element maps line
        pattern = NodeNodePattern;
      case 2 % Reading node node line
        fgetl(fid);fgetl(fid); % skip 2 lines
        [InstLocal2GlobalNodeNrsMap NoOfNodes] = ...
          ReadLocal2GlobalMap(fid,InstNameNumNumPattern);
        pattern = ElemElemPattern;
      case 3 % Reading element element line
        fgetl(fid);fgetl(fid); % skip 2 lines
        [InstLocal2GlobalElemNrsMap NoOfElems] = ...
          ReadLocal2GlobalMap(fid,InstNameNumNumPattern);
        break;
    end;
  end;
  tline = fgetl(fid);
end;
fclose(fid);

NoOfGroups = length(GrNr2GrNameMap);
%GroupYoungModulus = zeros(NoOfGroups,1);
GlobalElem2GrNrMap = zeros(NoOfElems,1);
GroupCenterPts = zeros(NoOfGroups,3);
GlobalNodeCoord = cell(1,2);
GlobalElem2GlobalNodeNbsMap = cell(1,2);
SetName2GlobalElemNrsMap = cell(1,2); % Global set name (instance name +
% delimiter + local set name), global element numbers

InstancePattern = '*(Instance),\s+name=([a-zA-Z0-9_-#]+),\s+part=([a-zA-Z0-9_-#]+)';
EndInstancePattern = '*(End Instance)';
NodePattern = '*(Node)';
ElemPattern = '*(Element)';
ElsetPattern = ['*(Elset),\s+elset=([a-zA-Z0-9_#]+),\s+internal[,\s+]' ...
                '*(generate|instance)?[=]*([a-zA-Z0-9_-#]+)?'];
Local2GlobalGrNamePattern = ['(Section),\s+elset=([a-zA-Z0-9_#]+),' ...
  '\s+material=([a-zA-Z0-9_-]+)'];
FreqPattern = '*(Frequency), eigensolver=';

Tokens = cell(1);
Tokens{1} = 'Instance';
Tokens{2} = 'Node';
Tokens{3} = 'Element';
Tokens{4} = 'Elset';
Tokens{5} = 'Section';
Tokens{6} = 'End Instance';
Tokens{7} = 'Frequency';
```

```
fid = fopen(AbaqusInputFile);
tline = fgetl(fid);
pattern = InstancePattern;
SetCount = 1;
TotNrOfElems = 0;
while ischar(tline) % if tline is array of characters
  [mat tok] = regexp(tline,pattern,'match','tokens');
  if ~isempty(tok)
    index = strmatch(tok{1}{1},Tokens,'exact');
    if ~isempty(index)
      %disp(['Case: ',num2str(index)])
      %disp(Tokens{index});
      switch index
        case 1 % Reading instance name
          InstanceName = upper(tok{1}{2});
          pattern = NodePattern;
          tline = fgetl(fid);
          NrOfElemsInInst = 0;
        case 2 % Reading node numbers with coordinates
          tline = fgetl(fid);
          str = regexp(tline,',','split');
          strn = strtrim(str);
          num = str2num(strn{1});
          while ~isempty(num)
            InstInd = strmatch(InstanceName, InstLocal2GlobalNodeNrsMap(:,1), 'exact');
            GlobalNodeInd = InstLocal2GlobalNodeNrsMap{InstInd,2}(num);
            GlobalNodeCoord{GlobalNodeInd,2} = ...
              [str2num(strn{2}) str2num(strn{3}) str2num(strn{4})];
            tline = fgetl(fid);
            str = regexp(tline,',','split');
            strn = strtrim(str);
            num = str2num(strn{1});
          end;
          pattern = [ElemPattern '|' EndInstancePattern];
        case 3 % Reading elements with node numbers
          tline = fgetl(fid);
          str = regexp(tline,',','split');
          strn = strtrim(str);
          num = str2num(strn{1});
          InstIndInElemMap = strmatch(InstanceName, InstLocal2GlobalElemNrsMap(:,1),'exact');
          InstIndInNodeMap = strmatch(InstanceName, InstLocal2GlobalNodeNrsMap(:,1),'exact');
          while ~isempty(num)
            NrOfElemsInInst = NrOfElemsInInst + 1;
            GlobalElemInd = InstLocal2GlobalElemNrsMap{InstIndInElemMap,2}(num);
            GlobalElem2GlobalNodeNbsMap{GlobalElemInd,2} = ...
              InstLocal2GlobalNodeNrsMap{InstIndInNodeMap,2}(cell2num(strn(2:end)));
            tline = fgetl(fid);
            str = regexp(tline,',','split');
            strn = strtrim(str);
            num = str2num(strn{1});
          end;

          pattern = [ElemPattern '|' ElsetPattern];
        case 4 % Reading elements in a set (both descriptive/generate)
          if size(tok{1},2)==4
            if strcmp(tok{1}{3},'') || strcmp(tok{1}{3},'instance')
              % Elset definition
              LocalSetName = tok{1}{2};
              if strcmp(tok{1}{3},'instance')
                InstanceName = upper(tok{1}{4});
              end;
              FullLocalSetName = [LocalSetName ...
                NameDelimiter InstanceName];
              tline = fgetl(fid);
              str = regexp(tline,',','split');
              strn = strtrim(str);
              % Match numbers in the string if the 1st token is
              % number
              Elems = [];
              while ~isempty(str2num(strn{1}))
                Elems = [Elems cell2num(strn)];
                tline = fgetl(fid);
```

```
186                 str = regexp(tline,',','split');
187                 strn = strtrim(str);
188               end;
189               ind=strmatch(InstanceName, InstLocal2GlobalElemNrsMap(:,1), 'exact');
190               SetName2GlobalElemNrsMap{SetCount,1}=FullLocalSetName;
191               SetName2GlobalElemNrsMap{SetCount,2}=InstLocal2GlobalElemNrsMap{ind,2}(Elems);
192               SetCount = SetCount + 1;
193             elseif strcmp(tok{1}{3},'generate') % Elset with generate
194               LocalSetName = tok{1}{2};
195               FullLocalSetName = [LocalSetName ...
196                 NameDelimiter InstanceName];
197               tline = fgetl(fid);
198               str = regexp(tline,'\d+','match');
199               tmp = cell2num(str);
200               Elems = tmp(1):tmp(3):tmp(2);

202               SetName2GlobalElemNrsMap{SetCount,1} = FullLocalSetName;
203               ind=strmatch(InstanceName, InstLocal2GlobalElemNrsMap(:,1), 'exact');
204               SetName2GlobalElemNrsMap{SetCount,2}=InstLocal2GlobalElemNrsMap{ind,2}(Elems);
205               SetCount = SetCount + 1;
206               tline = fgetl(fid);
207             end;
208           end;
209           pattern = [ElsetPattern '|' Local2GlobalGrNamePattern];
210         case 5 % Reading section definition
211           LocalSetName = tok{1}{2};
212           FullLocalSetName = [LocalSetName ...
213             NameDelimiter InstanceName];
214           ind1=strmatch(FullLocalSetName, SetName2GlobalElemNrsMap(:,1),'exact');

216           GlobalGroupName = tok{1}{3};
217           ind2 = strmatch(GlobalGroupName,GrNr2GrNameMap,'exact');
218           %tline
219           if ~isempty(ind2)
220             GlobalElem2GrNrMap(SetName2GlobalElemNrsMap{ind1,2}) = ...
221               ind2*ones(size(SetName2GlobalElemNrsMap{ind1,2}));
222             GlobalNodesInGroup ...
223                         = GlobalElem2GlobalNodeNbsMap(SetName2GlobalElemNrsMap{ind1,2},2);
224             NoOfElemsInGroup = length(GlobalNodesInGroup);
225             NoOfNodesInGroup = 0;
226             for j = 1:NoOfElemsInGroup
227               NodesInElem = GlobalNodesInGroup{j};
228               NoOfNodesInElem = length(NodesInElem);
229               for k=1:NoOfNodesInElem
230                 NoOfNodesInGroup = NoOfNodesInGroup + 1;
231                 GroupCenterPts(ind2,:) = GroupCenterPts(ind2,:) + ...
232                   GlobalNodeCoord{NodesInElem(k),2};
233               end;
234             end;
235             GroupCenterPts(ind2,:) = GroupCenterPts(ind2,:)/...
236               NoOfNodesInGroup;
237           end;
238           pattern = [Local2GlobalGrNamePattern '|' ElsetPattern ...
239             '|' InstancePattern ...
240             '|' EndInstancePattern];
241           tline = fgetl(fid);
242         case 6 % Reading end instance
243           pattern = [InstancePattern '|' ElsetPattern '|' FreqPattern];
244           TotNrOfElems = TotNrOfElems + NrOfElemsInInst;
245           tline = fgetl(fid);
246         case 7 % Reading total number of mode shapes in FEM model
247           tline = fgetl(fid);
248           str = regexp(tline,',','split');
249           strn = strtrim(str);
250           num = str2num(strn{1});
251           if ~isempty(num)
252             NoOfModes = num;
253           end
254           break
255       end; % switch
256     end; % if ~isempty(index)
257   else % if ~isempty(tok)
258     tline = fgetl(fid);
```

```
  end
end; % while
fclose(fid);

%% ================================================================
%% ==== SUPPLEMENTARY FUNCTIONS ===================================
%% ================================================================

%% ==== Convert a cell array of strings to a row vector of numbers ========
function arr = cell2num(cellArray)
% Avoid empty string caused by trailing commas:
if length(cellArray{end})==0
  cellArray=cellArray(1:end-1);
end

len = length(cellArray);
arr = zeros(1,len);
for i=1:len
  %   NextStr=cellArray{i};
  %   if ischar(NextStr)
  arr(i) = str2num(cellArray{i});
  %   end
end;


function [Map NoOfGlobals] = ReadLocal2GlobalMap(fid,InstNameNumNumPattern)
CurrentInstName = [];
GlobalNumsInInst = [];
MapInd = 0;
tline = fgetl(fid);
data = regexp(tline,InstNameNumNumPattern,'names');
while ~strcmp(tline,'')
  if ~isempty(data)
    if ~strcmp(CurrentInstName,data.name) % instance change
      if isempty(CurrentInstName)
        CurrentInstName = data.name;
        GlobalNumsInInst = [GlobalNumsInInst ...
          str2num(data.num2)];
      else
        MapInd = MapInd + 1;
        Map{MapInd,1} = CurrentInstName;
        Map{MapInd,2} = GlobalNumsInInst;
        CurrentInstName = data.name;
        GlobalNumsInInst = str2num(data.num2);
      end;
    else % filling the same instance with global element map
      GlobalNumsInInst = [GlobalNumsInInst ...
        str2num(data.num2)];
    end;
  end;
  tline = fgetl(fid);
  data = regexp(tline,InstNameNumNumPattern,'names');
end;
%if MapInd~=0
% For the last found instance
MapInd = MapInd + 1;
Map{MapInd,1} = CurrentInstName;
Map{MapInd,2} = GlobalNumsInInst;
NoOfGlobals = max(Map{MapInd,2});
%end
```

D.3.2 ComputeNewModes.m

Anropas av `fmincon` efter varje iteration. Skriver uppdaterade parametervärden till indatafil för Abaqus och anropar Abaqus för att räkna ut nya förutspådda moddata (modformer och -frekvenser).

```
function [OptConst,phiFEMcomp,nuFEMcomp] = ComputeNewModes(a,...
```

```
                                                    OptimValues,state,...
                                                    OptConst,StructUpdating)
% USAGE: OptConst = ComputeNewModes(a,OptimValues,state,setK0Mtx,OptConst)
%
% DESCRIPTION
% This function is called by fmincon after each iteration of the Newton
% trust region algorithm. It passes the updated decision parameters to
% Abaqus for computing new mode shapes, mode frequencies and stiffness
% matrices.
%
% INPUTS
% a                         = uppdated values of the decision parameters
% OptimValues               = a structure containing data from the current
%                             iteration thst we do not need and therefore
%                             will ignore. (See p. 9-19 in the Optimization
%                             Toolbox User's Guide)
% state                     = the current state or the algorithm, which we
%                             also do not need and therefore will ignore.
%                             (see p. 9-19 in the Optimization Toolbox
%                             User's Guide)
% OptConst                  = structure containing all necesary constants
%                             for the optimization algorithm,
%                             see InitAbaqusModel.m file.
%
% OUTPUTS
% OptConst = structure containing all necesary constants for the
%            optimization algorithm, see InitAbaqusModel.m file.

% Update parameters before Abaqus job is started
for i=1:size(StructUpdating,2)
    GroupIndex = OptConst.MaterialNr2GrNrMap(i);
    if GroupIndex~=0
       StructUpdating(i).Elastic = OptConst.P0(GroupIndex)*(1-a(GroupIndex));% OK?????
    end;
end;

% Compare to
% NoOfMaterials = size(StructUpdating,2);
% OptConst.MaterialNr2GrNrMap = zeros(NoOfMaterials,1)
% for i=1:NoOfMaterials
%     GroupIndex=find(ismember(GrNr2GrNameMap,StructUpdating(i).Name));
%     if ~isempty(GroupIndex)
%        StructUpdating(i).Elastic = OptConst.P0(GroupIndex);
%        OptConst.MaterialNr2GrNrMap(i) = GroupIndex;
%     end;
% end;

% Update Abaqus input file
UpdateAbaqusInpFile(OptConst.AbaqusInpFilName,...
                    OptConst.AbaqusModelPartFilName,...
                    OptConst.AbaqusHistPartFilName,...
                    StructUpdating);

% Run job in Abaqus
command = ['abaqus job=' OptConst.AbaqusInpFilName ' interactive'];
%t = clock;
%istatus = dos(command);
%display(['Abaqus job is finished in ' num2str(etime(clock,t)/60) ...
%        ' minutes with status: ' num2str(istatus)]);

% Read Abaqus output
phiFEMcomp(1) = NaN;
nanEntries = isnan(phiFEMcomp);
while ~isempty(find(nanEntries==1))
    t = clock;
    istatus = dos(command);
    display(['Abaqus job is finished in ' num2str(etime(clock,t)/60) ...
        ' minutes with status: ' num2str(istatus)]);
    [phiFEMcomp,nuFEMcomp] = ReadFreqDatFile(OptConst.AbaqusDatFilName,...
        OptConst.TotNrOfNodes,...
        OptConst.TotNrOfModes,...
        OptConst.NrOfDofsPerNode);
   % Kgr = ReadElementStiffMtxFile(OptConst.TotNrOfNodes,...
```

```
    %     OptConst.NrOfDofsPerNode,...
    %     OptConst.GlobalElem2GrNrMap,...
    %     OptConst.AbaqusElemStiffMtxFile,OptConst.SparseMtx);
    nanEntries = isnan(phiFEMcomp);
end;

% (We have no interest in the OptimValues and state input parameters in our
%  work, so we simply ignore them and stop here.)
```

D.3.3 divisors.m

Anropas av `SortRectMeshPts` för att hitta alla faktoriseringar av ett heltal i två andra heltal .

```
function nVec = divisors(N)
% INPUT
% N = an integer
%
% OUTPUT
% nVec = a column vector containg all integers dividing N, stored in
% increasing order.
PrimeFactors=factor(N);
PrimeFactorsPowers=FindPowers(PrimeFactors); % Defined below
nVec =  CombFactors(PrimeFactorsPowers);




function PrimeFactorsPowers=FindPowers(PrimeFactors)
% INPUTS
% PrimeFactors = a vector containing a sequence of prime factors in
%                increasing order
%
% OUTPUTS
% PrimeFactors = a 2 column vector with the first column containing the
%                uniqe prime factors in PrimeFactors and the second
%                column containing the number of occurences of that prime
%                factor in PrimeFactors.
ind=find(PrimeFactors==PrimeFactors(1));
pow=length(ind);
if pow==length(PrimeFactors)
  PrimeFactorsPowers= [PrimeFactors(1) pow];
else
  PrimeFactorsPowers=[PrimeFactors(1) pow
                      FindPowers(PrimeFactors(pow+1:end))];
end




function factors =  CombFactors(PrimeFactorsPowers)

if length(PrimeFactorsPowers(:,1))==1
  tmp=1;
else
  tmp=CombFactors(PrimeFactorsPowers(2:end,:));
end

NextPower=PrimeFactorsPowers(1,2);
factors=zeros((NextPower+1)*length(tmp),1);
for pp = 0:NextPower % The different powers
  factors(pp*length(tmp)+(1:length(tmp))) = ...
    [PrimeFactorsPowers(1,1).^pp*tmp ];
end

factors=sort(factors);
```

D.3.4 FEMupdating.m

Huvudprogram som sätter konstanter och startvärden, anropar `fmincon` för optimering av uppdateringsparametrar och sedan skriver ut resultat.

```
function [P0,Pupdated,fval,OptConst]= FemUpdating(varargin)

% USAGE: Eupdated = FemUpdating()
%
% OUTPUT
% P0       = column vector containing the initial Young's modulus of the
%            different (groups of) elements of the Abaqus FE model
% Pupdated = column vector containing the updated Young's modulus of the
%            different (groups of) elements of the Abaqus FE model
% fval     = final value of the objective function

%global aOld

%=========================================================================
%% Some parameters used for the communication with Abaqus ================
[OptConst,StructUpdating] = InitAbaqusModelGroupsOLD(varargin{:});


pLen = length(OptConst.CoarseMeshInd);  % == length(p) ~= length(a)
p0   = zeros(pLen,1); % Initial guess: Young modulus E=E0
%a0   = OptConst.NDamageFcts*p0;
P0   = OptConst.P0;
pMax=1-OptConst.Pmin(OptConst.CoarseMeshInd)./P0(OptConst.CoarseMeshInd).*ones(pLen,1);
pMin=1-OptConst.Pmax(OptConst.CoarseMeshInd)./P0(OptConst.CoarseMeshInd).*ones(pLen,1);

% Preset decision parameters
%aOld = zeros(OptConst.NrOfGroups,1); % Initial guess: Young modulus E=E0

%====== Run Abaqus to compute mode shapes, frequencies ==================
%OptConst = ComputeNewModes(a0,[],[],true,OptConst);
% OptConst will be updated with the value of global stiffness matrix

%====== Do the optimization =============================================
% Options for fmincon for
% * choosing the trust region algorithm
% * function name to call after each iteration
% * use gradient and Hessian output provided by ObjFun
options = optimset('Algorithm','trust-region-reflective',...
                   'GradObj','on',...
                   'Hessian','user-supplied', ...
                   'Display','iter');
% If GradObj is on, then objective function should return gradient
% vector in the second output argument.
% If Hessian is user-defined, then Hessian should be the final output
% of the objective function.

%% Call fmincon for FEM updating under the restriction aMin <= a <= aMax ==
% (Similar to the example on p. 2-12 with input arguments described on
%  p. 11-39 - 11-41 in the Optimization Toolbox User's Guide)

ObjectiveFunction = @(p) ObjFun(p,OptConst,StructUpdating);

[p,fval] = fmincon(ObjectiveFunction,p0,[],[],[],[],pMin,pMax,[],options);

a = OptConst.NDamageFcts*p;
Pupdated = P0.*(1-a); % Updated Young's modulus
display('Start    \    Found');
display([P0 Pupdated]);

%=========================================================================
```

D.3.5 GreedyPickLarge.m

Anropas av `PairModes` för att automatiskt para ihop förutspådda och uppmätta modformer parvis.

```
function indRC=GreedyPickLarge(A)
% Greedy algorithm for finding large elements in the matrix A without
% two of them being on the same row or column
%
% INPUTS
% A = rxc matrix
%
% OUPTUTS
% indRC = max(r,c)x2 matrix with each row containng the row- and column
%         index of one chosen element

[nofR,nofC]=size(A);
indR=1:nofR;
indC=1:nofC;
nofElements2pick=min(nofR,nofC);
indRC=zeros(nofElements2pick,2); % Reserve memory

ElementCtr=1; % Counter for next element to pick
while ( ElementCtr<=nofElements2pick  )
  [maxA,r,c]=rcMax(A);
  indRC(ElementCtr,:)=[indR(r),indC(c)];

  A = RemoveRowCol(A,r,c);
  indR = RemoveElem(indR,r);
  indC = RemoveElem(indC,c);

  ElementCtr=ElementCtr+1;
end


%% Test code
% A=rand(7,7)
% indRC=GreedyPickLarge(A)
```

D.3.6 FoxKapoor.m

För vinkelfrekvenser $\omega_m \overset{\text{def}}{=} 2\pi\nu_m$, räknas $N \times 1$-vektorer $\frac{\partial \omega_m^2}{\partial a_e}$ och $\frac{\partial \boldsymbol{\phi}_m^d}{\partial a_e}$ ut från Fox–Kapoor-forlerna (11):

$$\frac{\partial (2\pi\nu_m^{\text{FEM}})^2}{\partial a_e} = -\,\boldsymbol{\phi}_{d,m}^{\text{FEM}\,\text{T}} K_{0,e} \boldsymbol{\phi}_{d,m}^{\text{FEM}} \tag{12a}$$

$$\frac{\partial \boldsymbol{\phi}_{d,m}^{\text{FEM}}}{\partial a_e} = \sum_{q \neq m} \frac{\boldsymbol{\phi}_{d,q}^{\text{FEM}\,\text{T}} K_{0,e} \boldsymbol{\phi}_{d,m}^{\text{FEM}}}{(2\pi\nu_q^{\text{FEM}})^2 - (2\pi\nu_m^{\text{FEM}})^2} \boldsymbol{\phi}_{d,q}^{\text{FEM}} \tag{12b}$$

För att att undvika en for-loop implementeras ekvation (12b) som följer. För diagonalmatrisen

$$\Lambda_m \overset{\text{def}}{=} \begin{pmatrix} \lambda_{m,1} & 0 & \cdots & \cdots & 0 \\ 0 & \lambda_{m,2} & \ddots & & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \ddots & \lambda_{m,M-1} & 0 \\ 0 & \cdots & 0 & 0 & \lambda_{m,M} \end{pmatrix} \quad \text{med} \quad \lambda_{m,q} \overset{\text{def}}{=} \begin{cases} \frac{\boldsymbol{\phi}_q^{d\,\text{T}} K_{0,e} \boldsymbol{\phi}_m^d}{(2\pi\nu_q)^2 - (2\pi\nu_m)^2} & \text{för } q \neq m, \\ 0 & \text{för } q = m, \end{cases}$$

så följer det från (11b) att $\frac{\partial \boldsymbol{\phi}_m^d}{\partial a_e} = \sum_{q=1}^{M} \lambda_{m,q} \boldsymbol{\phi}_q^d$ är summan av kolumnvektorerna i

$$\Phi^d \Lambda_m \qquad \text{med} \qquad \Phi^d \overset{\text{def}}{=} \begin{pmatrix} \boldsymbol{\phi}_1^d & \cdots & \boldsymbol{\phi}_M^d \end{pmatrix} = \texttt{phiFEM}(:,:,\texttt{d}).$$

```
function [d_omegam2_d_ae,d_phim_d_ae] = FoxKapoor(phiFEMcomp,nuFEMcomp,K0gr,SparseMtx)
% USAGE: [d_omegam2_d_ae,d_phim_d_ae] = FoxKapoor(SparseMtx)
%
% DESCRIPTION
% Computes the formulas (12) and (13) of Fox and Kapoor in
% "Damage assessment..."
%
% INPUTS
% phiFEMcomp = NxMxD vector of mode shape column vectors.
% nuFEMcomp  = Mx1 vector of mode frequencies
% K_0e   = NxNxE array of NxN element stiffness matrices
%
% OUTPUTS
% d_omegam2_d_ae = MxE-matrix
% d_phim_d_ae    = NxMxDxE-array


[N,M,D]=size(phiFEMcomp);
if SparseMtx
  E=length(K0gr);   % Number of (groups of) elements
else
  E=length(K0gr(1,1,:));   % Number of (groups of) elements
end

if D>1
  warning(['The Fox-Kapoor formulas are probably not yet correctly ' ...
           'implemented for vibration modes that are not pure ' ...
           'x-, y- or z-direction bending modes.'])
end
d_omegam2_d_ae = zeros(M,E,D); % Temporary D-dependence.
                               % See the last rows below.
d_phim_d_ae = zeros(N,M,D,E);

for e=1:E
  for m=1:M
    for d=1:D
      % Compute (12) in "Damage assessment..."
      if SparseMtx
        K0grMat=K0gr{e};
      else
        K0grMat=K0gr(:,:,e);
      end
      d_omegam2_d_ae(m,e,d) = -phiFEMcomp(:,m,d).'*K0grMat*phiFEMcomp(:,m,d);
      % Compute (13) in "Damage assessment..." without for loop:
      Lambda_mDiagonal = phiFEMcomp(:,:,d).'*K0grMat*phiFEMcomp(:,m,d) ...
                         ./ ( (2*pi*nuFEMcomp).^2 -(2*pi*nuFEMcomp(m))^2 );
      Lambda_mDiagonal(m)=0;
      Lambda_m = diag(Lambda_mDiagonal);
      % sum of column vectors
      d_phim_d_ae(:,m,d,e)=sum( (phiFEMcomp(:,:,d)*Lambda_m).' ).';
    end
    % Quick and dirty handling of potentially different Fox-Kapoor
    % right-hand sides for different d:
    ind=find( abs(d_omegam2_d_ae(m,e,:))>1000*eps );
    v=abs(d_omegam2_d_ae(m,e,ind));
    if ( (max(v)-min(v))>1000*eps )
      error(['Mode frequencies not correctly computed ' ...
             'in the multivariate case'])
    else
      d_omegam2_d_ae(m,e,:)=mean(d_omegam2_d_ae(m,e,ind));
    end
  end
end
```

D.3.7 InitAbaqusModelGroupsOLD.m

Sätter konstanter samt startvärden på uppdateringsparametrar.

```
function [OptConst,StructUpdating] = InitAbaqusModelGroupsOLD(AbaqusInpFilName, ...
```

```
                                    MasterFemNodes,MasterFemDofs,...
                                    MasterMeasDofs,...
                                    nuMeas,phiMeas,...
                                    GroupNames,...
                                    Pmin,Pmax,P0,...
                                    CoarseMeshInd,...
                                    MeasX,MeasY,nX,nY,MeasDirection,...
                                    NrOfDofsPerNode,...
                                    RefNode,RefDof,...
                                    SparseMtx,...
                                    ShapeResType,WeightingStrategy,...
                                    ModePairingMethod,...
                                    ModeScalingMethod,...
                                    W,...
                                    PlotModeShapes)

OptConst = struct('AbaqusInpFilName',0,        ... % Abaqus input file name
                  'MasterInd',0,               ... % indicies of meas points in entire modal
                                               ... % shape should correspond to meas
                  'MasterFemDofs',0,           ... % needed for extraction from FEM model
                  'MasterMeasDofs',0,          ...
                  'RefNodeIndex',0,            ... % node corresponding to excitation
                  'NrOfGroups',0,              ... % number of groups to be updated
                  'NrOfDofsPerNode',0,         ... % in the FEM model (for preallocation)
                  'Pmin',0,                    ... % P min constrain
                  'Pmax',0,                    ... % P max constrain
                  'P0',0,                      ... % initial value (guess)
                  'nuMeas',0,                  ... % eigen freqs from meas
                  'phiMeas',0,                 ... % eigen vectors from meas
                  'CoarseMeshInd',0,           ... % in FemUpdating.m
                  'SparseMtx',0,               ...
                  'ShapeResType',0,            ...
                  'WeightingStrategy',0,       ...
                  'ModePairingMethod',0,       ...
                  'ModeScalingMethod',0,       ...
                  'Weights',0,                 ...
                  'TotNrOfNodes',0,            ... % is read from Abaqus inp file (for
                                               ... % preallocation of mode shape and
                                               ... % frequencies vector)
                  'TotNrOfModes',0,            ... % is read from Abaqus dat file
                                               ... % (for preallocation)
                  'AbaqusDatFilName',0,        ... % created based on inp file name
                  'AbaqusModelPartFilName',0,  ... % created from Abaqus inp file
                  'AbaqusHistPartFilName',0,   ... % created from Abaqus inp file
                  'MaterialNr2GrNrMap',0,      ...
                  'NDamageFcts',0,             ...
                  'K0gr',0,                    ... % loaded after 1st run of FEM model TEST
                                               ... % that from element we produce the same
                                               ... % answer
                  'gridX',0,                   ...
                  'gridY',0,                   ...
                  'MeasDirection',0,           ...
                  'PlotModeShapes',0);

if nargin==16 % use default values
    disp('Processing of 14 arguments for FemUpdating');
    OptConst.NrOfDofsPerNode   = 6;
    OptConst.SparseMtx         = true;
    OptConst.ShapeResType      = 'l_2';
    OptConst.WeightingStrategy = 'none';
    OptConst.ModePairingMethod = 'MAC';
    OptConst.ModeScalingMethod = 'SameSign';
    OptConst.PlotModeShapes    = true;
    OptConst.Weights = Weights(size(phiMeas,2),...
                size(phiMeas,2)*length(MasterFemNodes)*length(MasterFemDofs),...
                OptConst.WeightingStrategy);
    RefNode = 0; RefDof = 0;
%elseif nXrgin==18 % do not plot mode shapes
elseif nargin<16 || (nargin>15 && nargin<26)
    error('Not enought number of parameters!')
else
    OptConst.NrOfDofsPerNode   = NrOfDofsPerNode;
    OptConst.SparseMtx         = SparseMtx;
```

```
    OptConst.ShapeResType      = ShapeResType;
    OptConst.WeightingStrategy = WeightingStrategy;
    OptConst.ModePairingMethod = ModePairingMethod;
    OptConst.ModeScalingMethod = ModeScalingMethod;
    OptConst.PlotModeShapes    = PlotModeShapes;
    OptConst.Weights           = W;
end

OptConst.AbaqusInpFilName = AbaqusInpFilName;
% FEM mode numbers corresponding to measured modes. Can be defined
% automatically by MAC criteria against measurement modes.
%OptConst.MasterModes = [7 8 9 10 12 13 16 17 18 19 20 21 22]; % FRF, undamaged
OptConst.MasterFemDofs = MasterFemDofs;
% x(=1), y(=2), z(=3)
OptConst.MasterMeasDofs = MasterMeasDofs;
OptConst.nuMeas     = nuMeas;
OptConst.phiMeas    = phiMeas;

N = length(MasterFemNodes);
D = length(MasterFemDofs);
OptConst.MasterInd = zeros(N*D,1);
k = 1;
for i=MasterFemNodes'
    OptConst.MasterInd(k:k+D-1) = (i-1)*OptConst.NrOfDofsPerNode+OptConst.MasterFemDofs;
    k = k+D;
end;

if RefDof~=0
    % Choose nPlateExcFrc as reference degree of freedom for normalizing
    % phiFEM and phiMeas in the function ModeShapeResid
    nPlateExcFrc = (find(MasterFemNodes==RefNode)-1)*length(OptConst.MasterFemDofs)+...
                find(OptConst.MasterFemDofs==RefDof); % index of the excitation
    OptConst.RefNodeIndex=nPlateExcFrc;
else
    % Will make ModeShapeResid call ChooseRefAcc for choosing nRef:
    OptConst.RefNodeIndex=0;
end

OptConst.NrOfGroups = length(GroupNames);
% if length(MasterFemNodes)~=OptConst.NrOfGroups
%     error('%s\n%s%d%s','Number of nodes on which the predicted mode shapes', ...
%           'will be indentified should be ',OptConst.NrOfGroups,'!');
% end;

OptConst.Pmin = Pmin; %Pa (Young modulus lower bound)
OptConst.Pmax = Pmax; %Pa (Young modulus upper bound)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initialization of the updating parameters and settings their bounds %%%%%
% For the rectangular measurement grid P0 is initialized by strips with the
% same Young modulus. You should know how the groups are defined in the
% Abaqus model to make a good choice of initial parameters! For the plate
% the groups were defined from upper left to lower right corners in columns
% and therefore the way to initialize the updating parameters! But this
% should be redefined if the group definition is different from the
% described above. Of course, one can set all group initial parameter value
% to the same constant if it is appropriate.
OptConst.P0 = P0;

OptConst.CoarseMeshInd = CoarseMeshInd;

OptConst.gridX = linspace(0,MeasX,nX); % along rect area
OptConst.gridY = linspace(0,MeasY,nY);   % across rect area
OptConst.MeasDirection = MeasDirection;

% NOTE! Here you define number and Name (see *.inp file) for each group
% Number of the group associated with a particular material will be the
% same as index of this material in the GroupNames cell array
GrNr2GrNameMap = cell(1,OptConst.NrOfGroups);
for i=1:OptConst.NrOfGroups
    GrNr2GrNameMap{i} = GroupNames{i};
end;

% We assume that the Name of the .dat and .mtx files coincides with the
```

```
% Name of the Abaqus input file, if not change the code accordingly
PointIndex = find(OptConst.AbaqusInpFilName=='.');
FileName = OptConst.AbaqusInpFilName(1:PointIndex-1);
OptConst.AbaqusDatFilName       = [FileName,'.DAT'];
OptConst.AbaqusModelPartFilName = [FileName,'__PartI'];
OptConst.AbaqusHistPartFilName  = [FileName,'__PartII'];
AbaqusInitHistPartFilName = [FileName,'-INIT__PartII'];
AbaqusUpdatingPartFilName = [FileName,'_UPDATE'];
AbaqusElemStiffMtxFilName = upper([FileName,'.MTX']);

CutMaterialPartFromInpFile(OptConst.AbaqusInpFilName,...
                           OptConst.AbaqusModelPartFilName,...
                           AbaqusUpdatingPartFilName,...
                           OptConst.AbaqusHistPartFilName,...
                           AbaqusInitHistPartFilName);

StructUpdating = ReadStructMaterial(AbaqusUpdatingPartFilName);
NoOfMaterials = size(StructUpdating,2);
OptConst.MaterialNr2GrNrMap = zeros(NoOfMaterials,1);

for i=1:NoOfMaterials
    GroupIndex=find(ismember(GrNr2GrNameMap,StructUpdating(i).Name));
    if ~isempty(GroupIndex)
       StructUpdating(i).Elastic = OptConst.P0(GroupIndex);
       OptConst.MaterialNr2GrNrMap(i) = GroupIndex;
    end;
end;

UpdateAbaqusInpFile(OptConst.AbaqusInpFilName,...
                    OptConst.AbaqusModelPartFilName,...
                    AbaqusInitHistPartFilName,... % NOTE! Element stiffness
                    StructUpdating);              % matrices are computed!

% Run job in Abaqus
% Preiteration run is done since it produces element stiffness matrices
% that are needed to be produced just one time. From these matrices the
% group stiffness matrices are built. It is also more easy to read the
% total number of nodes from the .dat file. Total number of nodes is
% required later on when reading the modal data for preallocation of
% vectors for the mode shapes.
% See if it is possible to avoid this.

command = ['abaqus job=' OptConst.AbaqusInpFilName ' interactive'];

t = clock;
istatus = dos(command);
display(['Abaqus job is finished in ' num2str(etime(clock,t)/60) ...
         ' minutes with status: ' num2str(istatus)]);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%% The constants below are read from the Abaqus files %%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Extract number of modes, number of nodes, mapping between global element
% and group number and group center points
 [OptConst.TotNrOfModes,OptConst.TotNrOfNodes,...
 GlobalElem2GrNrMap,GroupCenterPnts] = BuildGlobalElemNr2GrNrLokup_NEW2(...
                                       OptConst.AbaqusDatFilName,...
                                       OptConst.AbaqusInpFilName,...
                                       GrNr2GrNameMap);

OptConst.NDamageFcts=InterpolFunctions(GroupCenterPnts(:,1:2),...
                                    OptConst.CoarseMeshInd);

OptConst.K0gr = ReadElementStiffMtxFile(OptConst.TotNrOfNodes,...
                                     OptConst.NrOfDofsPerNode,...
                                     GlobalElem2GrNrMap,...
                                     AbaqusElemStiffMtxFilName,...
                                     OptConst.SparseMtx);
```

D.3.8 InputParameters.m

Sätter lämpliaga värden på indataparametrar för `FEMupdating`.

```
AbaqusInpFilName = 'Model-e-3-updated.inp';
%%%%%%%%%%%%%%%%%%%%%% MasterNodes %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% NOTE! Order in which nodes appear in MasterNodes vector should be
% exactly the same as the order for nodes in phiMeas vector in which
% measurements were done.
MasterFemNodes = [56 119 181 243 332 ...
                  52 115 177 239 324 ...
                  48 111 173 235 318 ...
                  44 107 169 231 312 ...
                  40 103 165 227 306 ...
                  36 99 161 223 300 ...
                  32 95 157 219 294 ...
                  28 91 153 215 288 ...
                  24 87 149 211 282 ...
                  20 83 145 207 276 ...
                  16 79 141 203 270 ...
                  12 75 137 199 264 ...
                  8 71 133 195 258]';
MasterFemDofs  = 3;
MasterMeasDofs = 3;

ModeShapeFile = 'TEST\MODES_FRF\ModeShapes13ModesNoDamage.txt'; % undamaged
ModeFreqFile  = 'TEST\MODES_FRF\Frequencies13ModesNoDamage.txt'; %undamaged
phiMeas = dlmread(ModeShapeFile);
nuMeas  = dlmread(ModeFreqFile);
nuMeas  = nuMeas(:,1);

NrOfDofsPerNode = 6;

RefNode = 8; % node number corresponding to measurement point closest to the
             % excitation force  (shaker)
RefDof  = 3; % excitation dof (1=X, 2=Y, 3=Z direction)

NrOfGroups = 65;
GroupNames = cell(1,NrOfGroups);
ConcreteStr = 'concrete-';
for i=1:NrOfGroups
    GroupNames{i} = strcat(ConcreteStr,num2str(i));
end;

SparseMtx         = true;
ShapeResType      = 'l_2';
WeightingStrategy = 'none';
ModePairingMethod = 'MAC';
ModeScalingMethod = 'SameSign';

W = Weights(size(phiMeas,2),...
            size(phiMeas,2)*length(MasterFemNodes)*length(MasterFemDofs),...
            WeightingStrategy);

CoarseMeshInd = [1 3 5 11 13 15 21 23 25 31 33 ...
                 35 41 43 45 51 53 55 61 63 65];

Pmin = 10*10^9*ones(NrOfGroups,1); %Pa (Young modulus lower bound)
Pmax = 40*10^9*ones(NrOfGroups,1); %Pa (Young modulus upper bound)
% Make initial guess of parameters in order in which groups appear in
% GrNr2GrNameMap

MeasY  = 26.6; % cm
MeasX = 79.98; % cm
nY = 5;  % number of measurement points in each row
nX = 13; % number of measurement rows
MeasDirection = 'AlongY'; % 'AlongX' or 'AlongY'
% make guess with the same values in each measurement row (column in meas grid!)
P0 = repmat(35+ 10*(rand(1,nX)-0.5),nY,1)*10^9;
P0 = P0(:);

PlotModeShapes = true;
```

D.3.9 InterpolFunctions.m

Producerar "tältfunktioner" som kan användas för att med linjär interpolation räkna ut värden för uppdateringsparametrar i punkterna `FineMesh` från uppdaterade värden i punkterna `CoarseMesh`.

```
function L=InterpolFunctions(MeshPts,CoarseMeshInd)
% USAGE: L=InterpolFunctions(FineMesh,CoarseMeshRowColInd)
%
% INPUTS
% FineMesh            = a 3-dimensional array with FineMesh(r,c,:1) and
%                       FineMesh(r,c,:2) being the x- and the y-coordinates
%                       of one point in the mesh
% CoarseMeshRowColInd = length 2 cell array with RowsInCoarseMesh and
%                       ColsInCoarseMesh being vectors containing the
%                       index sets for he first and second coordinate of
%                       FineMesh, respectively. This gives the coarse mesh
%                       for each point in which a tent function is computed.


%% ==== If no input, then make a default choice for demonstration/testing ==
if nargin==0
  DemoMode=true;
  NofRows=11;
  NofCols=31;
  NofRows=3 % DEBUG-OPTION!!!
  NofCols=7 % DEBUG-OPTION!!!
  D=2;
  SigmaMesh=0.1; % Standard deviation for random deviation of mesh points
                 % from the integers.
              SigmaMesh=0
  FineMesh=zeros(NofRows,NofCols,D);
  for rr=1:NofRows
    for cc=1:NofCols
      FineMesh(rr,cc,:)=[cc,rr];
    end
  end

  FineMesh=FineMesh+SigmaMesh*randn(size(FineMesh));
  NofPtsFineMesh=NofRows*NofCols;
  MeshPts=reshape(FineMesh,NofPtsFineMesh,D);
  ind=randperm(NofPtsFineMesh);
  MeshPts=MeshPts(ind,:);

  RowsInCoarseInd=[1:5:11];
  ColsInCoarseInd=[1:5:31];
  RowsInCoarseInd=[1:NofRows] % DEBUG-OPTION!!!
  ColsInCoarseInd=[1:NofCols] % DEBUG-OPTION!!!
  PtInCoarseGrid=repmat(false,NofRows,NofCols);
  PtInCoarseGrid(RowsInCoarseInd,ColsInCoarseInd)=true;
  PtInCoarseGrid=PtInCoarseGrid(:);   % Reshape to column vector.
  PtInCoarseGrid=PtInCoarseGrid(ind); % The same reordering as for MeshPts.
  CoarseMeshInd=find(PtInCoarseGrid);
else
  DemoMode=false;
end

%% == Sort mesh points into a (small deviation from) a rectangular mesh  ==
[RectMesh,RectMeshInd,RowsInCoarseMesh,ColsInCoarseMesh] ...
                                  = SortRectMeshPts(MeshPts,CoarseMeshInd);
[NofRows,NofCols,D]=size(RectMesh);
NofPtsFineMesh=NofRows*NofCols;

%% ==== Plot the numbering of the centerpoints ============================
if DemoMode
  FigNr=0;
  FigNr=FigNr+1;
  figure(FigNr);
```

```
63    clf

65    FineGridColour=colour('green');
66    CoarseGridColour=colour('red');

68    xCoords=RectMesh(:,:,1);
69    xMin=min(xCoords(:));
70    xMax=max(xCoords(:));
71    yCoords=RectMesh(:,:,2);
72    yMin=min(yCoords(:));
73    yMax=max(yCoords(:));
74    delta=0.15;
75    axis([xMin-delta xMax+delta yMin-delta yMax+delta])


78    IsInCoarseGrid=repmat([false],NofRows,NofCols);
79    IsInCoarseGrid(RowsInCoarseMesh,ColsInCoarseMesh)=true;
80    for cc=1:NofCols
81      for rr=1:NofRows
82        xy=RectMesh(rr,cc,:);
83        x=xy(1);
84        y=xy(2);
85        if IsInCoarseGrid(rr,cc)
86          text(x,y,int2str(RectMeshInd(rr,cc)),'BackgroundColor',CoarseGridColour)
87        else
88          text(x,y,int2str(RectMeshInd(rr,cc)),'BackgroundColor',FineGridColour)
89        end
90        hold on
91      end
92    end
93    hold off
94    xlabel('x')
95    ylabel('y')
96    title(['Center point positions (red for coarse mesh, \sigma_{mesh}=' ...
97           num2str(SigmaMesh) ') numbered in the order that they appear ' ...
98           'in the input parameter MeshPts.'])
99    grid on
100 end

102 %% ==== Create tent functions ================================================
103 NofCMrows=length(RowsInCoarseMesh);
104 NofCMcols=length(ColsInCoarseMesh);

106 L=zeros(NofPtsFineMesh,NofCMrows*NofCMcols);
107 %TentFctNr=0;
108 for cInd=1:NofCMcols
109   cc=ColsInCoarseMesh(cInd);
110   for rInd=1:NofCMrows
111     %TentFctNr=TentFctNr+1;
112     rr=RowsInCoarseMesh(rInd);
113     TentFct=zeros(NofRows,NofCols);
114     TentFct(rr,cc)=1;

116     ind = max(find(RowsInCoarseMesh<rr));
117     if length(ind)==0
118       PrevRow=rr;
119     else
120       PrevRow = RowsInCoarseMesh(ind);
121     end
122     ind = min(find(RowsInCoarseMesh>rr));
123     if length(ind)==0
124       NextRow=rr;
125     else
126       NextRow = RowsInCoarseMesh(ind);
127     end
128     ind = max(find(ColsInCoarseMesh<cc));
129     if length(ind)==0
130       PrevCol=cc;
131     else
132       PrevCol = ColsInCoarseMesh(ind);
133     end
134     ind = min(find(ColsInCoarseMesh>cc));
135     if length(ind)==0
```

```
136       NextCol=cc;
137     else
138       NextCol = ColsInCoarseMesh(ind);
139     end


142     %% ==== Create all 1-8 "tent walls" ====================================
143     % Lower left quadrant:
144     rInd=PrevRow:rr;
145     cInd=PrevCol:cc;
146     if  ( (length(rInd)>1) && (length(cInd)>1) )
147       x = RectMesh(rInd,cInd,1);
148       y = RectMesh(rInd,cInd,2);
149       PointsInPlane=[x(end,end) x(1,1)   x(1,end)
150                      y(end,end) y(1,1)   y(1,end)
151                      1          0        0];
152       TentWall1=zCoordsForPlane(PointsInPlane,x,y);
153       PointsInPlane=[x(end,end) x(end,1) x(1,1)
154                      y(end,end) y(end,1) y(1,1)
155                      1          0        0];
156       TentWall2=zCoordsForPlane(PointsInPlane,x,y);
157       TentFct(rInd,cInd)=min(TentWall1,TentWall2);
158     end

160     % Upper left quadrant:
161     rInd=rr:NextRow;
162     cInd=PrevCol:cc;
163     if  ( (length(rInd)>1) && (length(cInd)>1) )
164       x = RectMesh(rInd,cInd,1);
165       y = RectMesh(rInd,cInd,2);
166       PointsInPlane=[x(end,1) x(1,end) x(1,1)
167                      y(end,1) y(1,end) y(1,1)
168                      0          1        0     ];
169       TentWall1=zCoordsForPlane(PointsInPlane,x,y);
170       PointsInPlane=[x(end,1) x(1,end) x(end,end)
171                      y(end,1) y(1,end) y(end,end)
172                      0          1        0     ];
173       TentWall2=zCoordsForPlane(PointsInPlane,x,y);
174       TentFct(rInd,cInd)=min(TentWall1,TentWall2);
175     end

177     % Upper right
178     rInd=rr:NextRow;
179     cInd=cc:NextCol;
180     if  ( (length(rInd)>1) && (length(cInd)>1) )
181       x = RectMesh(rInd,cInd,1);
182       y = RectMesh(rInd,cInd,2);
183       PointsInPlane=[x(1,1) x(end,1) x(end,end)
184                      y(1,1) y(end,1) y(end,end)
185                      1          0        0];
186       TentWall1=zCoordsForPlane(PointsInPlane,x,y);
187       PointsInPlane=[x(1,1) x(end,end) x(1,end)
188                      y(1,1) y(end,end) y(1,end)
189                      1          0        0];
190       TentWall2=zCoordsForPlane(PointsInPlane,x,y);
191       TentFct(rInd,cInd)=min(TentWall1,TentWall2);
192     end
193     % lower right
194     rInd=PrevRow:rr;
195     cInd=cc:NextCol;
196     if  ( (length(rInd)>1) && (length(cInd)>1) )
197       x = RectMesh(rInd,cInd,1);
198       y = RectMesh(rInd,cInd,2);
199       PointsInPlane=[x(end,1) x(1,1) x(1,end)
200                      y(end,1) y(1,1) y(1,end)
201                      1          0        0       ];
202       TentWall1=zCoordsForPlane(PointsInPlane,x,y);
203       PointsInPlane=[x(end,1) x(1,end) x(end,end)
204                      y(end,1) y(1,end) y(end,end)
205                      1          0        0       ];
206       TentWall2=zCoordsForPlane(PointsInPlane,x,y);
207       TentFct(rInd,cInd)=min(TentWall1,TentWall2);
208     end
```

```
    if DemoMode
        FigNr=FigNr+1;
        figure(FigNr);
        x = RectMesh(:,:,1);
        y = RectMesh(:,:,2);
        %mesh(1:NofCols,1:NofRows,L);
        surf(x,y,TentFct);
        xlabel('x')
        ylabel('y')
        title(['Tent function with center point (' ...
            num2str(RectMesh(rr,cc,1))  ',' ...
            num2str(RectMesh(rr,cc,2))  ')'])
    end
    % Finally, reorder the tent function points in the original order given
    % by the index set RectMeshInd and the relation
    %
    %   RectMesh(r,c,d) = MeshPts(RectMeshInd(r,c),d)
    for rrr=1:NofRows
      for ccc=1:NofCols
        % Fine mesh point number for the current tent function:
        CurrentTentFctFineMeshPtNr=RectMeshInd(rr,cc);
        TentFctNr=find(CoarseMeshInd==CurrentTentFctFineMeshPtNr);

        L(RectMeshInd(rrr,ccc),TentFctNr)=TentFct(rrr,ccc);
      end
    end
  end
end
```

D.3.10 Jacobian.m

Räknar ut Jacobianen $J_r(\boldsymbol{a})$ som definieras i (9).

$$(J_{\boldsymbol{r}}(\boldsymbol{a}))_{k,l} \stackrel{\text{def}}{=} \frac{\partial r_k}{\partial a_l}$$

```
function Jr = Jacobian(phiFEM,phiMeas,nuMeas,phiFEMcomp,nuFEMcomp,OptConst,MasterModes)
% USAGE: Jr = Jacobian(phiFEM,phiMeas,nuMeas,OptConst)
%
% DESCRIPTION
% Computes Jacobian matrix J_r with (J_r)_{k,l} = (d r_k)/(d a_l)
%
% INPUTS

% phiFEM = NxMxD vector of mode shape column vectors (FEM output).
% nuFEM  = Mx1 vector of mode frequencies (FEM output).
% ShapeResType = The type of shape residual used.
% nRef   = index of reference column in phiFEM
% phiMeas  = NxMxD vector of mode frequencies (OMA output).
% nuMeas  = Mx1 vector of mode frequencies (OMA output).
% K0   = NxNxE array of NxN element stiffness matrices
%
% OUTPUTS
% J_r = RxE-matrix with R=M(DN+1)

%global Kgr

[N,M,D]=size(phiFEM);
%[N,M,D]=size(phiFEMcomp);
if OptConst.SparseMtx
  E=length(OptConst.K0gr);
else
  E=length(OptConst.K0gr(1,1,:));   % Number of (groups of) elements
end

% Compute the Fox-Kapoor formulas (12) and (13) in "Damage assessment..."
[d_lambda_m_d_ae,d_phim_d_ae] = FoxKapoor(phiFEMcomp,nuFEMcomp,OptConst.K0gr, ...
                                          OptConst.SparseMtx);
%%% Reducton the output from Fox-Kapoor to %%%%%%%%%%%%
```

```
d_lambda_m_d_ae = d_lambda_m_d_ae(MasterModes,:,:); % MxExD
d_phim_d_ae     = d_phim_d_ae(OptConst.MasterInd,MasterModes,:,:); % (N,M,D,E)

JrUpper = diag( (2*pi*nuMeas).^(-2) ) * d_lambda_m_d_ae; % MxE-matrix

%% Compute mode shape residual derivatives ================================
JrLower=zeros(N,M,D,E); % Will be reshaped to N*M*DxE-matrix after the loop
for dd=1:D
  for ee=1:E
    if strcmp(OptConst.ShapeResType,'diff')
      for mm=1:M
        JrLower(:,mm,dd,ee) ...
          = ( d_phim_d_ae(:,mm,dd,ee)*phiFEM(nRef,mm,dd) ...
              - phiFEM(:,mm,dd)*d_phim_d_ae(nRef,mm,dd,ee) ) ...
            ./ (phiFEM(nRef,mm,dd).^2);
      end
    elseif strcmp(OptConst.ShapeResType,'l_inf')
      for mm=1:M
        [InfNormPhi,nMaxFEM] = max(abs(phiFEM(:,mm,dd)));
        % Differentiate abs(phiFEM(nMaxFEM,mm,dd)) using the chain rule:
        DiffInfNormPhi = d_phim_d_ae( nMaxFEM(1),mm,dd,ee) ...
                         *sign(phiFEM(nMaxFEM(1),mm,dd));
        JrLower(:,mm,dd,ee) ...
          = ( d_phim_d_ae(:,mm,dd,ee )*InfNormPhi ...
              - phiFEM(:,mm,dd)*DiffInfNormPhi ...
            ) ./ (InfNormPhi.^2);
      end
    elseif strcmp(OptConst.ShapeResType,'l_2')
      for mm=1:M
        normPHIfem = norm(phiFEM(:,mm,dd));
        JrLower(:,mm,dd,ee) = d_phim_d_ae(:,mm,dd,ee)./normPHIfem ...
                              - (d_phim_d_ae(:,mm,dd,ee)' ...
                                 *phiFEM(:,mm,dd))/(normPHIfem.^3) ...
                                *phiFEM(:,mm,dd);
      end
    else
      for mm=1:M
        A = phiFEM(:,mm,dd).'*phiMeas(:,mm,dd);
        B = d_phim_d_ae(:,mm,dd,ee).'*phiMeas(:,mm,dd);
        C = phiFEM(:,mm,dd).'*phiFEM(:,mm,dd);
        DD = phiMeas(:,mm,dd).'*phiMeas(:,mm,dd);
        F = d_phim_d_ae(:,mm,dd,ee).'*phiFEM(:,mm,dd);
        if strcmp(OptConst.ShapeResType,'MSF')
          JrLower(:,mm,dd,ee) = ( (2*A*F-B*C)/C*phiFEM(:,mm,dd) ...
                                  - A*d_phim_d_ae(:,mm,dd,ee))...
                                ./(C*phiMeas(:,mm,dd));
        elseif strcmp(OptConst.ShapeResType,'MAC')
          for nn=1:N
            JrLower(nn,mm,dd,ee)= 2*A*DD*(A*F-B*DD)/(C.^2*DD);
          end
        else
          error('Incorrect input ''ShapeResType''')
        end
      end
    end
  end
end

JrLower=reshape(JrLower,N*M*D,E);

%% Assemble the full Jacobian ============================================
Jr=[JrUpper
    JrLower];
```

D.3.11 ModeShapeResid

Räknar ut residualen (6).

```
function [r_s,nRef,phiRelOMA,phiRelFEM] ...
                              = ModeShapeResid(phiFEM,phiOMA,method,nRef)
```

```
% Computes different residuals describing the size of the difference
% between the simulated and measured mode shapes phiFEM and phiOMA.
%
% USAGE: r_s = ModeShapeResid(phiFEM,phiOMA,method,nRef)
%
% INPUT
% phiFEM = NxMxD array with length N FEM-simulated mode shapes.
% phiOMA = NxMxD array with length N measured mode shapes.
% method = a string telling which method to use for comparison
%          'diff':          difference of mode shapes normalized to have element
%                           number nRef equal to one.
%          'l_2':  difference of mode shapes normalized to have element
%                           number nRef equal to one.
%          'MSF' :          difference of mode shapes normalized to have element
%                           number nRef equal to one
%          'MAC' :          The sine squared of the angles between the column
%                           vectors in phiFEM and phiOMA, respectively.
%                           (Scalar, contrary top the previous ones.)
%
% nRef   = optional input specifyinng which node point that will be used
%          for normalization before comparing phiFEM and phiOMA.
%          If not specified of if nRef=0, ChooseRefAcc is called for
%          an automatic choice of nRef.
%
% OUTPUT
% r_s       =
% nRef      = same at the input nRef, if present and nonzero, and otherwise
%             chosen by calling the function ChooseRefAcc.
% phiRelOMA = the resulting normalized phiOMA that is used if the input
%             method is 'diff'
% phiRelFEM = the resulting normalized phiOMA that is used if the input
%             method is 'diff'

[N,M,D]=size(phiFEM);
r_s = zeros(N,M,D);

%%% ==== First check signs ================================================
% phiFEM=SameSign(phiFEM,phiOMA);

%% ==== First, do normalization ==========================================
if (nargin<3 || nRef == 0)
  % Choose the reference node point that will be used for normalization.
  [phiRelOMA,phiRelFEM,nRef] = ChooseRefAcc(phiOMA,phiFEM);
else
  phiRelOMA = zeros(N,M,D);
  phiRelFEM = zeros(N,M,D);
  for mm=1:M
    for dd=1:D
       if ( strcmp(method,'diff') && (min(abs([phiFEM(nRef,mm,:);phiOMA(nRef,mm,:)]))<1e-8 ))
           error('uh oh...')
       end
      phiRelFEM(:,mm,:) = phiFEM(:,mm,:)/phiFEM(nRef,mm,:);
      phiRelOMA(:,mm,:) = phiOMA(:,mm,:)/phiOMA(nRef,mm,:);
    end
  end;
end

%% ==== Next, compute the residual =======================================
if strcmp(method,'diff')
  r_s = phiRelFEM - phiRelOMA;
elseif strcmp(method,'l_inf')
  for mm=1:M
    for dd=1:D
      r_s(:,mm,dd) = phiFEM(:,mm,dd)./max(abs(phiFEM(:,mm,dd))) ...
                   -  phiOMA(:,mm,dd)./max(abs(phiOMA(:,mm,dd)));
    end
  end
elseif strcmp(method,'l_2')
  for dd = 1:D
    for mm = 1:M
      normPHIfem = norm(phiFEM(:,mm,dd));
      normPHIoma = norm(phiOMA(:,mm,dd));
      r_s(:,mm,dd) = phiFEM(:,mm,dd)./normPHIfem ...
```

```
                    - phiOMA(:,mm,dd)./normPHIoma;
    end
  end
elseif strcmp(method,'MSF')
  % The normalized phiRelOMA and phiRelFEM not needed
  r_s=zeros(N,M,D);
  for mm=1:M
    for dd=1:D
      r_s(:,mm,dd) = ( phiOMA(:,mm,dd) ...
                     - phiFEM(:,mm,dd).'*phiOMA(:,mm,dd)*phiFEM(:,mm,dd) ...
                       ./norm(phiFEM(:,mm,dd))^2);
    end
  end
elseif strcmp(method,'MAC')
  r_s=zeros(1,M,D);
  for mm=1:M
    for dd=1:D
      r_s(1,mm,dd) = 1 - ( phiFEM(:,mm,dd).'*phiOMA(:,mm,d) )^2 ...
                       /(  phiFEM(:,mm,dd).'*phiFEM(:,mm,d) ...
                         *phiOMA(:,mm,dd).'*phiOMA(:,mm,d) );
    end
  end
else
  error('Incorrect input ''method''')
end
```

D.3.12 ObjFun.m

Anropas av `fmincon` för att räkna ut värden, derivator och andraderivator av målfunktionen med formlerna i Avsnitt D.2.

```
function [f,Gradf,HfApprox]=ObjFun(p,OptConst,StructUpdating)
% USAGE: [f,Gradf,HfApprox]=ObjFun(p,OptConst)
%
% DESCRIPTION
%
% INPUTS
% a       = Ex1-vector with decision parameters.  For element/group nr e,
%           the updated physical property X(e) relates to its initial value
%           via the multiplication X(e)=X_0(e)(1-a(e))
% OptConst = structure containing all necesary constants for the
%            optimization algorithm, see InitAbaqusModel.m file.
%
% OUTPUTS
% f        = objective function
% Gradf    = gradient of f
% HfApprox = approximation of the Hessian of f

%global aOld
%global phiFEMcomp
%global nuFEMcomp

%first time is zero but should be of right size
a = OptConst.NDamageFcts*p;

disp([ 'a= [ ' num2str(a') ' ]' ]);
Prounded=round((OptConst.P0.*(1-a))'/10^8)/10;
disp([ 'P= [ ' num2str(Prounded) ' ]' ])
% Call Abaqus for computing new phiFEM, nuFEM and K_0e:
if norm(a)~=0
  [OptConst,phiFEMcomp,nuFEMcomp] = ComputeNewModes(a,[],[],OptConst,StructUpdating);
else
  [phiFEMcomp,nuFEMcomp] = ReadFreqDatFile(OptConst.AbaqusDa ...tFilName,...
    OptConst.TotNrOfNodes,...
    OptConst.TotNrOfModes,...
    OptConst.NrOfDofsPerNode);
end;

phiFEM = phiFEMcomp(OptConst.MasterInd,:);
```

```
if ~strcmp(OptConst.ModePairingMethod,'none')
    [MasterModes,phiFEM,nuFEM,phiMeas,nuMeas] = PairModes(phiFEM,...
                          nuFEMcomp,OptConst.phiMeas,OptConst.nuMeas,...
                          OptConst.ModePairingMethod);
else
    MasterModes = 1:length(OptConst.nuMeas);
    phiFEM = phiFEM(:,MasterModes);
    nuFEM  = nuFEMcomp(MasterModes);
end;

if ~strcmp(OptConst.ModeScalingMethod,'none')
    phiMeas = ScaleModes(phiFEM,OptConst.phiMeas,OptConst.ModeScalingMethod);
end;

display(['nu = [ ' num2str(nuFEM') ']']);
%% Compute the residuals and the objective function ======================
%% Make sure that phiFEM and phiMeas "have the right sign"
% % Probabbly should be changed to MSF ???

r_f = (nuFEM.^2 - OptConst.nuMeas.^2)./OptConst.nuMeas.^2;
r_s = ModeShapeResid(phiFEM,phiMeas,OptConst.ShapeResType,...
    OptConst.RefNodeIndex);
r_f = r_f(:); % Make column vector
r_s = r_s(:); % Make column vector
r=[r_f(:) ; r_s(:)]; % The residual

%Redefine Weights in order to satisfy r!!!!

%[r,w]=ResidAndWeights(r_f,r_s,WeightingStrategy);
W=diag(OptConst.Weights);             % Diagonal weight matrix
f = sum( r.^2.*OptConst.Weights )/2;  % The objective function

%% Compute the gradient and the Hessian of f ============================
Jra = Jacobian(phiFEM,phiMeas,OptConst.nuMeas, ...
               phiFEMcomp,nuFEMcomp,OptConst,MasterModes); % an RxE-matrix with R=M(DN+1)

Jrp=Jra*OptConst.NDamageFcts;
Gradf = Jrp.'*W*r;
disp(['ObjFun  : ' num2str(f)]);
disp(['Grad Fox:' num2str(Gradf.')]);
HfApprox = Jrp.'*W*Jrp;
if size(HfApprox,1)==1
    disp(['Hess Fox:' num2str(HfApprox.')]);
else
    disp(['Hess Fox (1,1) (1,2): ' num2str(HfApprox(1,1)) ', ' num2str(HfApprox(1,2))]);
    disp(['Hess Fox (2,1) (2,2): ' num2str(HfApprox(2,1)) ', ' num2str(HfApprox(2,2))]);
end;

figure(OptConst.NrOfGroups)
Pupdated = OptConst.P0.*(1-a);
len = length(r_f);
wF = OptConst.Weights(1:len);
lenS = length(phiFEM);
wS = OptConst.Weights(len+1:lenS:end);
PlotUpdatingParams(OptConst.P0,Pupdated,OptConst.gridX,OptConst.gridY,wF',wS',f);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if OptConst.PlotModeShapes
    PlotModeShapes(OptConst,MasterModes,phiFEM,nuFEM,phiMeas,nuMeas);
end

%% Import current mode shapes, mode frequencies and stiffness matrices ====
% The function fmincon calls ComputeNewModes after each iteration of the
% Newton trust region algorithm  for running Abaqus and copy some chosen
% to the global variable struct_abaqus, from which we now will need the
% last computed mode shapes, their mode frequencies and the local
% stiffness matrix for each (group of) elements.
%
% In the current version of this software, we only compare the bending mode
% shapes in z-direction

function phiMeas = ScaleModes(phiFEM,phiMeas,ModeScalingMethod)
```

```
if strcmp(ModeScalingMethod,'SameSign')
    phiMeas=SameSign(phiFEM,phiMeas);
elseif strcmp(ModeScalingMethod,'MSF')
    n = size(phiMeas,2);
    MSF = zeros(n,1);
    for i=1:n
        MSF(i) = phiFEM(:,i).'*phiMeas(:,i)/(phiMeas(:,i).'*phiMeas(:,i));
        phiMeas(:,i) = MSF(i)*phiMeas(:,i);
    end;
end;
```

D.3.13 PairModes.m

Anropas av `ObjFun` för att para ihop förutspådda och uppmätta modformer.

```
function [BestMatchingFEMind,phiFEM,nuFEM,phiMeas,nuMeas] = ...
                    PairModes(phiFEM,nuFEM,phiMeas,nuMeas,ModePairingMethod)
% (Uppdated October 34 2013 by Niklas)

DEBUG = true

nofObsModes = size(phiMeas,2); % Number of observed modes in measurements
nofPredModes = size(phiFEM,2); % Number of predicted modes from FE model

if strcmp(ModePairingMethod,'MAC')

  %% ==== MAC criteria for pairing of mode shapes ============================
  MACt = zeros(nofPredModes,nofObsModes); % Table of MAC values
  for pp=1:nofPredModes
    for oo=1:nofObsModes
      MACt(pp,oo) = MAC(phiFEM(:,pp),phiMeas(:,oo));  % Defined below
    end;
  end;

  indRC=GreedyPickLarge(MACt);
  %% Now sort the picked pairs so that the observed modes (columns)
  %  keep their original order and the predicted modes (rows) are
  %  reordered accordingly
  [dummy,ind]=sort(indRC(:,2));  %  B r motsvara OptConst.MasterModes
  if DEBUG
    MACt=MACt
    indRC=indRC
    ind=ind
  end
  BestMatchingFEMind=indRC(ind,1);
  phiFEM = phiFEM(:,BestMatchingFEMind);
  nuFEM  = nuFEM(   BestMatchingFEMind);

  if DEBUG
    disp(['Permutation indices for FEM modeshapes: [' num2str(BestMatchingFEMind(:).') ']'])
    disp('Should be similar to the folowing manual choice done by Natalia:')
    disp(['OptConst.MasterModes = [7 8 9 10 12 13 16 17 18 19 20 21 22]; % FRF, undamaged'])
  end

  %% Mode pairing with respect to modeshapes done.
  %  The observed mode shapes (phiMEAS) are now sorted after increasing
  %  frequency (nuMeas), but the same might not hold for the predicted
  %  mode shapes

  % Code will follow here for warning messages when the predicted modes not
  % comes in increasing frequency order.
  for ii = 1:(length(ind)-1)
    if ind(ii)>ind(ii)+1
      warning('xxx')
    end
  end

%   %% OLD VERSION:
%   [maxx lookupTable] = max(MACt); % lookupTable contains the row numbers of
%                                   % the positions where the maximum
```

```
%                                   % elements of each column in MACt are
%                                   % placed
%   %lookupTable(4) = 4;
%   len = length(lookupTable);
%   %rem = [];
%   for k=1:len
%     ind = find(lookupTable==lookupTable(k));
%     if length(ind)>1
%       % If more than one observed mode share the same "closest" predicted
%       % mode shape
%       tmpMAC = MACt(:,ind(1));
%       tmpMAC(lookupTable(ind(1))) = 0;
%       [mm1 ii1] = maxx(tmpMAC);  % Find SECOND best match
%       if mm1>0.5 && lookupTable(ind(1))>ii1 % Why second condition????
%         lookupTable(ind(1)) = ii1;
%         %             elseif mm1<=0.5 && lookupTable(ind(1))>ii1
%         %                 rem = [rem k];
%       end;
%       tmpMAC = MACt(:,ind(2));
%       tmpMAC(lookupTable(ind(2))) = 0;
%       [mm2 ii2] = maxx(tmpMAC);
%       %[mm2 ii2] = max(MACt(MACt(:,ind(2))~=maxi(ind(2)),ind(2)));
%       if mm2>0.5 && lookupTable(ind(2))<ii2 % Why secod condition?????????
%         lookupTable(ind(2)) = ii2;
%         %             elseif mm2<=0.5 && lookupTable(ind(2))<ii2
%         %                 rem = [rem k];
%       end
%     end;
%   end;
%   %warning('ComputeNewModeShapes.m: Remove change in lookup table!!!');
%   phiFEM = phiFEM(:,lookupTable);
%   nuFEM = nuFEM(lookupTable);
end;


%% ==== Modal assurance criterion for vectors u and v ======================
function MACuv = MAC(u,v)
MACuv = abs(u'*v)^2 / ( u'*u * v'*v );
```

D.3.14 PlotModeShapes.m

Anropas av `ObjFun` för att plotta modformer om konstanten `OptConst.PlotModeShapes` är satt till `true`.

```
function PlotModeShapes(OptConst,MasterModes,phiFEM,nuFEM,phiMeas,nuMeas)

% Is anB scaling of mode shapes required here??

ModeFigNum = 1;
if ~isempty(find(findobj('type','figure')==ModeFigNum,1))
    clf(ModeFigNum),
end;
[X,Y] = meshgrid(OptConst.gridX,OptConst.gridY);
% X,Y - matrices of length(OptConst.gridB) x length(OptConst.gridA) size
nX = length(OptConst.gridX);
nY = length(OptConst.gridY);
NrOfModes = length(MasterModes);
NrOfCols = 3;
NrOfRows = ceil(NrOfModes/NrOfCols);
NrOfDofs = length(OptConst.MasterFemDofs);
if strcmp(OptConst.MeasDirection,'AnongX')
    n1 = nX; n2 = nY;
else %'AlongY'
    n1 = nY; n2 = nX;
end;
figure(ModeFigNum);
for k=1:NrOfModes
    subplot(NrOfRows,NrOfCols,k);
    XYZfem=zeros(size(X,1),size(X,2),3);
```

```
    XYZmea=zeros(size(X,1),size(X,2),3);
    for kk=1:NrOfDofs
        XYZfem(:,:,OptConst.MasterMeasDofs(kk)) = ...
            reshape(phiFEM(kk:NrOfDofs:end,k),n1,n2);
        XYZmea(:,:,OptConst.MasterMeasDofs(kk)) = ...
            reshape(phiMeas(kk:NrOfDofs:end,k),n1,n2);
    end;
    %ZZfem = reshape(OptConst.phiFEM(:,k),nB,nA);
    surf(X+XYZfem(:,:,1),Y+XYZfem(:,:,2),XYZfem(:,:,3)); hold on
    %ZZmea = reshape(OptConst.phiMeas(:,k),nB,nA);
    surf(X+XYZmea(:,:,1),Y+XYZmea(:,:,2),XYZmea(:,:,3)); hold on
    title(['FEM mode ' num2str(MasterModes(k)) ': ' ...
           num2str(nuFEM(k)) ' Hz (Meas ' ...
           num2str(nuMeas(k)) ' Hz)']);
end;
```

D.3.15 PlotUpdatingParams.m

Anropas av `ObjFun` för att plotta aktuellt värde på uppdateringsparametrar.

```
function PlotUpdatingParams(P0,P,gridX,gridY,wF,wS,f)
% P = vector of length 21

P=P(:); % Make column vector
P=reshape(P,length(gridY),length(gridX));
% is it still ok if the number of groups less than the number of meas
% points???
X = gridX;
Y = gridY;
pcolor2(X,Y,P);
colorbar

nn=0;
for nX = 1:length(X)
    for nY = 1:length(Y)
      nn=nn+1;
      text(X(nX),Y(nY)+2, ['P0=' num2str( round(P0(nn)/10^8)/10 )], ...
                          'HorizontalAlignment','center' )
      text(X(nX),Y(nY), ['P=' num2str( round(P(nn)/10^8)/10 )], ...
                         'HorizontalAlignment','center' )
    end
end

title(['P for {\it w_f}=[' num2str(wF) ...
        '], {\it w_s}=[' num2str(wS)...
        '], obj fun=' num2str(f)] )



function pcolor2(X,Y,P)
P= [P P(:,end)
    P(end,:) P(end,end)];
X=X(:);
deltaX=X(2)-X(1);
X=[X-deltaX/2;X(end)+deltaX/2];
Y=Y(:);
deltaY=Y(2)-Y(1);
Y=[Y-deltaY/2;Y(end)+deltaY/2];
%Y=flipud(Y);
pcolor(X,Y,P);
```

D.3.16 rcMax.m

Anropas av `GreedyPickLarge` för att hitta maximalt element i en matris, samt rad- och kolumnindex för detta.

```
function [maxA,r,c]=rcMax(A);
% Find the largest element in the matrix A, as well as its row- and column
% index of

[rows,cols]=size(A);
if ( rows==1 )
  r=1;
  [maxA,c]=max(A);
elseif ( cols==1 )
  c=1;
  [maxA,r]=max(A);
else
  [colMax,RowInds]=max(A); % Max of each column and the row index for each
  [maxA,c]=max(colMax);
  r=RowInds(c);
end

%% Test code
% A=rand(4,4)
% [maxA,r,c]=rcMax(A)
% A=rand(1,4)
% [maxA,r,c]=rcMax(A)
% A=rand(4,1)
% [maxA,r,c]=rcMax(A)
```

D.3.17 ReadElementStiffMtxFile.m

Anropas av `InitAbaqusModelGroupsOLD` för att läsa in styvhetsmatrisen K_0 från utdatafil från Abaqus.

```
function Ke = ReadElementStiffMtxFile(nofNodes,nofDofs,lookup,file,SparseMtx)

% Note we cannot compose the global stiffness matrix by using .mtx file,
% since it could happen that the element set associated with this file is
% not complete, i.e. not cover the whole structure. Check for that the
% Abaqus .inp file for the definition of ELEMENT MATRIX OUTPUT.
% If we need the global stiffness and mass matrices it is a good idea to
% retrieve them in separate files by saying this in the .inp file.

% Example:
% Ke = ReadElementStiffMtxFile(1500,6,[ones(365,1);2],'Matrix.txt',1);
% file = 'Matrix.txt';
% SparseMtx = 1;
% nofNodes = 1500; % number of global nodes
% nofDofs = 6; % Assume each node in the FE model has 6 dofs
% nofGroups = 2;
% lookup = ones(366,1);
% lookup(366) = 2;

ElementNumberPattern   = '**\s*(ELEMENT NUMBER)\s+(\d+)';
UserElemNodesPattern   = '*(USER ELEMENT, NODES)=\s+(\d+)';
ElementNodesPattern    = '**\s*(ELEMENT NODES)';
StiffMatrixPattern     = '*\s*(MATRIX,TYPE=STIFFNESS)';
IntNumberPattern       = '\d+';
MatrixElementsPattern  = '[+-]?(\d+\.\d+|\d+\.|\.\d+|\d+)([eE][+-]?\d+)?';

Tokens = cell(1);
Tokens{1} = 'ELEMENT NUMBER';
Tokens{2} = 'USER ELEMENT, NODES';
Tokens{3} = 'ELEMENT NODES';
Tokens{4} = 'MATRIX,TYPE=STIFFNESS';

nofGroups = max(lookup); % no. of groups

KNofRows  = nofNodes*nofDofs; % Nr of rows in the stiffness matrix
if SparseMtx
    Ke = cell(nofGroups,1);
    for gg=1:nofGroups
```

```
        Ke{gg} = sparse(KNofRows,KNofRows);
    end
else
    Ke = zeros(KNofRows,KNofRows,nofGroups);
end


fid   = fopen(file);
tline = fgetl(fid); % Read next line from file

pattern = ElementNumberPattern;
while ischar(tline) % if tline is array of characters
    [str tok] = regexp(tline,pattern,'match','tokens');
    if ~isempty(str)
        %disp(str)
        index = strmatch(tok{1}{1},Tokens,'exact');
        switch index
            case 1 % Reading element number line
                ElemNumber = str2num(tok{1}{2});
                GrNr = lookup(ElemNumber);
                pattern = UserElemNodesPattern;
            case 2 % Reading element nodes number line
                nofNodesInElem = str2num(tok{1}{2});
                pattern = ElementNodesPattern;
            case 3 % Reading nodes in the current element
                NodesInElem = [];
                while ( length(NodesInElem) < nofNodesInElem )
                    tline = fgetl(fid);
                    %disp(tline)
                    ElemNodesStr=regexp(tline,IntNumberPattern,'match');
                    NodesInElem = [ NodesInElem cell2num(ElemNodesStr)];
                end
                tline = fgetl(fid);
                str=regexp(tline,',','split');
                str = strtrim(str);
                dofNrs = cell2num(str);
                LocalNofDofs = length(dofNrs);
                pattern = StiffMatrixPattern;
            case 4 % Reading element stiffness matrix
                KeNofRows = nofNodesInElem*LocalNofDofs;
                % Nr rows in the element stiffness matrix
                KeLoc = zeros(KeNofRows); % Element stiffnes matrix
                % === Read the elements in and below the diagonal of KeLoc
                for RowNr = 1:KeNofRows
                    MatrixElements=[];
                    while ( length(MatrixElements) < RowNr )
                     tline = fgetl(fid);
                     MatrElemStr=regexp(tline,MatrixElementsPattern,'match');
                     MatrixElements = [ MatrixElements cell2num(MatrElemStr)];
                    end
                    KeLoc(RowNr,1:RowNr)=MatrixElements;
                end
                KeLoc = Triang2SymmMtx(KeLoc);
                Dofs  = LocalInd2GlobalInd(NodesInElem,dofNrs);
                % Assemble the current element stiffnes matrix at the right
                % place in the current element group stiffness matrix
                if GrNr~=0
                    if SparseMtx
                        Ke{GrNr}(Dofs,Dofs) = Ke{GrNr}(Dofs,Dofs) + KeLoc;
                    else
                        Ke(Dofs,Dofs,GrNr) = Ke(Dofs,Dofs,GrNr) + KeLoc;
                    end
                end;
                pattern = ElementNumberPattern;
        end;

    end;
    tline = fgetl(fid); % Read next line from file
end;

fclose(fid);

%% ========================================================================
```

```
%% ==== SUPPLEMENTARY FUNCTIONS ================================================
%% ==============================================================================

%% ==== Convert a cell array of strings to a row vector of numbers ==========
function arr = cell2num(cellArray)
% Avoid empty string caused by trailing commas:
if length(cellArray{end})==0
    cellArray=cellArray(1:end-1);
end

len = length(cellArray);
arr = zeros(1,len);
for i=1:len
    arr(i) = str2num(cellArray{i});
end;

%% ==== Translate from local to global stiffnes matrix row/column index =====
function Dofs = LocalInd2GlobalInd(NodesInElem,dofNrs)

LocalNofDofs = length(dofNrs);
nofNodesInElem = length(NodesInElem);

offset  = LocalNofDofs*(NodesInElem-1);
offsets = repmat(offset,LocalNofDofs,1);
tmp = offsets + repmat(dofNrs(:),1,nofNodesInElem);
Dofs = tmp(:);
```

D.3.18 ReadFreqDatFile.m

Anropas av `ComputeNewModes.m` och `ObjFun.m` för att läsa in modfrekvenser från utdatafil från Abaqus.

```
function [phiFEMcomp,nuFEMcomp] = ReadFreqDatFile(file,...
                                                  TotNrOfNodes,...
                                                  TotNrOfModes,...
                                                  NrOfDofsPerNode)
% INPUT:
% file  = name for the Abaqus ".dat" file with eigenfrequencies and mode shapes
% TotNrOfNodes = number of nodes in the entire FE model
% TotNrOfModes = number of modes found for the FE model
% NrOfDofsPerNode = number of dofs for each node in the FE model

% OUTPUT:
% nuFEMcomp  = TotNrOfModes number of eigenfrequencies for the entire FE model
% phiFEMcomp = TotNrOfModes number of mode shapes corresponding to nuFEMcomp and
%              defined for the predefined set in the Abaqus '*.dat' file

phiFEMcomp = zeros(TotNrOfNodes*NrOfDofsPerNode,TotNrOfModes);
nuFEMcomp  = zeros(TotNrOfModes,1);

EigenvalueOutputPattern = '**\s*(E I G E N V A L U E    O U T P U T)\s+';
EigenvalueNumberPattern = '**\s*(E I G E N V A L U E    N U M B E R)\s+(\d+)';

Tokens = cell(1);
Tokens{1} = 'E I G E N V A L U E    O U T P U T';
Tokens{2} = 'E I G E N V A L U E    N U M B E R';

fid = fopen(file);
tline = fgetl(fid);
%ModeNumber = 0; % mode shape number counter

pattern = EigenvalueOutputPattern;

while ischar(tline) % if tline is array of characters
    [str tok] = regexp(tline,pattern,'match','tokens');
    if ~isempty(str)
        %disp(str)
        index = find(strcmp(Tokens,tok{1}{1}));
        switch index
```

```
            case 1 % Reading eigenvalue output
                % skip 5 lines
                fgetl(fid);fgetl(fid);fgetl(fid);fgetl(fid);fgetl(fid);
                str = fgetl(fid);
                while ~isempty(str)
                    data = textscan(str,'%s%f64%f64%f64%f64%f64'); % read formatted data
                    nuFEMcomp(sscanf(cell2mat(data{1}),'%u')) = data{4}; % frequency in
                                                                         % cycles/time
                    str = fgetl(fid)
                end;
                pattern = EigenvalueNumberPattern;
            case 2 % Reading eigenvalue number
                ModeNumber = str2num(tok{1}{2});
                % skip 14 lines
                fgetl(fid);fgetl(fid);fgetl(fid);fgetl(fid);fgetl(fid);
                fgetl(fid);fgetl(fid);fgetl(fid);fgetl(fid);fgetl(fid);
                fgetl(fid);fgetl(fid);fgetl(fid);fgetl(fid);
                data = strread(fgetl(fid));
                ind = 1;
                while ~isempty(data)
                    phiFEMcomp(ind:ind+NrOfDofsPerNode-1,ModeNumber) = ...
                        data(2:NrOfDofsPerNode+1);
                    ind = ind + NrOfDofsPerNode;
                    data = strread(fgetl(fid));
                end;
                %ModeNumber = ModeNumber + 1;
            case 0
                break;
        end;
    end;
    tline = fgetl(fid);
end;
fclose(fid);
```

D.3.19 ReadStructMaterial.m

Anropas av `InitAbaqusModelGroupsOLD.m` för att läsa in uppdateringsparametrar från Abaqus.

```
function StructMaterial = ReadStructMaterial(AbaqusMaterialPartFilName)
% USAGE: StructMaterial = ReadStructMaterial('AbaqusMaterialSectionFile.inp')

StructMaterial = struct('Name',0,'Density',0,'Elastic',0,'Poisson',0);

MaterialPattern = '*(Material),\s+name=([a-zA-Z0-9_-#]+)';
ElasticPattern = '*(Elastic)';
DensityPattern = '*(Density)';

Tokens{1} = 'Material';
Tokens{2} = 'Density';
Tokens{3} = 'Elastic';

fid = fopen(AbaqusMaterialPartFilName);
tline = fgetl(fid);
pattern = MaterialPattern;

MaterialIndex = 0;

while ischar(tline) % if tline is array of characters
    %disp(tline)
    %if strcmp(tline,'*Frequency, eigensolver=Lanczos, acoustic coupling=on, normalization=↩
        mass')
    [mat tok] = regexp(tline,pattern,'match','tokens');
    if ~isempty(tok)
        index = strmatch(tok{1}{1},Tokens,'exact');
        if ~isempty(index)
            switch index
                case 1 % Reading material properties
                    MaterialName = tok{1}{2};
                    MaterialIndex = MaterialIndex + 1;
                    StructMaterial(MaterialIndex).Name = MaterialName;
```

```
                    pattern = DensityPattern;
                case 2 % Reading density value
                    tline = fgetl(fid);
                    str = regexp(tline,',','split');
                    strn = strtrim(str);
                    StructMaterial(MaterialIndex).Density = str2num(strn{1});
                    pattern = ElasticPattern;
                case 3 % Reading elastic properties
                    tline = fgetl(fid);
                    str = regexp(tline,',','split');
                    strn = strtrim(str);
                    StructMaterial(MaterialIndex).Elastic = str2num(strn{1});
                    StructMaterial(MaterialIndex).Poisson = str2num(strn{2});
                    pattern = MaterialPattern;
            end;
        end;
    end;
    tline = fgetl(fid);
end;
```

D.3.20 RemoveElem.m

Anropas av `GreedyPickLarge` för att plocka bort ett element från en vektor.

```
function vR = RemoveElem(v,n)
% Removes element number n from input vector/matrix v and returns the
% remaining elements in the column vector vR

v=v(:); % Make column vector
vLen=length(v);

if( (1<=n) && (n<=vLen) )
  vR=v([1:n-1 n+1:vLen]);
else
  error('Input argoments n not in the right range')
end



%% Test code
% v=[ 1 2 3 4 5]
% v1=RemoveElem(v,1)
% v2=RemoveElem(v,2)
% v3=RemoveElem(v,3)
% v4=RemoveElem(v,4)
% v5=RemoveElem(v,5)
```

D.3.21 RemoveRowCol.m

Anropas av `GreedyPickLarge` för att plocka bort en given rad och kolumn från en matris.

```
function Ar = RemoveRowCol(A,r,c)
% Removes row number r and column number c from the matrix A
[nofR,nofC]=size(A);
if( (1<=r) && (r<=nofR) && (1<=c) && (c<=nofC) )
  Ar=A([1:r-1 r+1:nofR],[1:c-1 c+1:nofC]);
else
  error('Input argoments r and c not in the right range')
end



%% Test code
% A=[ 11 12 13 14
%     21 22 23 24
```

```
%     31 32 33 34
%     41 42 43 44]
% A11=RemoveRowCol(A,1,1)
% A12=RemoveRowCol(A,1,2)
% A13=RemoveRowCol(A,1,3)
% A14=RemoveRowCol(A,1,4)
% A21=RemoveRowCol(A,2,1)
% A22=RemoveRowCol(A,2,2)
% A23=RemoveRowCol(A,2,3)
% A24=RemoveRowCol(A,2,4)
% A31=RemoveRowCol(A,3,1)
% A32=RemoveRowCol(A,3,2)
% A33=RemoveRowCol(A,3,3)
% A34=RemoveRowCol(A,3,4)
% A41=RemoveRowCol(A,4,1)
% A42=RemoveRowCol(A,4,2)
% A43=RemoveRowCol(A,4,3)
% A44=RemoveRowCol(A,4,4)
```

D.3.22 SameSign.m

Anropas av `ObjFun.m` för normera och välja samam tecken på förutspådda och uppmätta vibrationsmoder.

```
function [phiOMA SignFactor]=SameSign(phiFEM,phiOMA)
M = size(phiOMA,2);
MSF = zeros(M,1);
for i=1:M
   MSF(i) = (phiFEM(:,i).'*phiOMA(:,i))/(phiOMA(:,i).'*phiOMA(:,i));
end;
phiOMA = phiOMA*diag(MSF);
SignFactor = MSF;
```

D.3.23 SortRectMeshPts.m

Anropas av `InterpolFunctions.m` för utifrån koordinater sortera en mängd mätpunkter till ett rektangulärt punktnät.

```
function [RectMesh,RectMeshInd,RowsInCoarseMesh,ColsInCoarseMesh] ...
                                  = SortRectMeshPts(MeshPts,CoarseMeshInd)
% INPUT
% MeshPts       = Nx2-matrix with each row containing the x- and
%                 y-coordinate of one point in the mesh
% CoarseMeshInd = vector contaning the row indices of the points in the
%                 mesh that belongs to the coarse mesh.
%
% OUTPUTS
% RectMesh            = the mesh points sorted into a rectangular array
%                       RectMesh(m,n,d), with smaller m (or n)
%                       corresponding to smaller y (or x) and with
%                       d=1 and d=2 giving the x and the y-coordinate,
%                       respectively.
% RectMeshInd         = index set such that
%                       RectMesh(m,n,d) = MeshPts(RectMeshInd(m,n),d)
% RowsInCoarseMesh    = vector telling what rows that are used for the
%                       coarse mesh
% ColsInCoarseMesh    = vector telling what columns that are used for the
%                       coarse mesh

%% Reorder the mesh points in increasing x-coordinate order
[MeshPtsSortedX,indX]=sort(MeshPts(:,1));
MeshPtsSortedX=MeshPts(indX,:);

%% Find the number of rows such that the mesh point positions is a small
```

```
%  deviation from a rectangular mesh of size NofRows x NofCols:
NofPts=length(MeshPts(:,1));
PossibleNofRows=divisors(NofPts); % All possible column lengths
PossibleNofRows=PossibleNofRows(2:end-1).'; % Look for rectangular mesh

% Default if no rectangular mesh is found:
   RectMeshFound=false;

for NofRows = PossibleNofRows;
  NofCols = NofPts/NofRows;

  % Reshape before sorting:
  MeshPtsSortedXY=reshape(MeshPtsSortedX,NofRows,NofCols,2);
  % Sort each column in increasing y-coordinate order
  [MeshPtsSortedXY(:,:,2),indY]=sort(MeshPtsSortedXY(:,:,2));
  for cc=1:NofCols
    MeshPtsSortedXY(:,cc,1)=MeshPtsSortedXY(indY(:,cc),cc,1);
  end
  % Check if the current value of NofRows gives a separation of rows & cols
  NotRectMesh = false;
  for rr=1:NofRows-1
    % Check if each row is located below next row:
    if max(MeshPtsSortedXY(rr,:,2))>=min(MeshPtsSortedXY(rr+1,:,2))
      NotRectMesh = true;
    end
  end
  for cc=1:NofCols-1
    % Check if each column is located to the left of next column:
    if max(MeshPtsSortedXY(:,cc))>=min(MeshPtsSortedXY(:,cc+1))
      NotRectMesh = true;
    end
  end

  if ~NotRectMesh
    RectMeshFound=true;
    RectMesh=MeshPtsSortedXY;
    RectMeshInd=reshape(indX,NofRows,NofCols);
    for cc = 1:NofCols
      RectMeshInd(:,cc) = RectMeshInd(indY(:,cc),cc);
    end
  end
end

if ~RectMeshFound
  figure(666)
  clf
  for nn=1:length(MeshPts)
    plot(MeshPts(nn,1),MeshPts(nn,2),'r*')
    hold on
  end
  hold off
  error(['The input mesh points are not positioned with small ' ...
         'deviations from a rectangular mesh'])
end

%% ==== Find rows and columns used for the coarse mesh ====================
[NofRows,NofCols]=size(RectMeshInd);
blah=zeros(NofRows,NofCols);
for nn=1:length(CoarseMeshInd(:))
  ind=find(RectMeshInd==CoarseMeshInd(nn));
  blah(ind)=1;
end
% If there are two different nonzero rows or columns then the rank >1:
if rank(blah)>1
  error('not possible to build tent functions for this mesh')
end
RowsInCoarseMesh = find( sum(blah,2)~=0 );
ColsInCoarseMesh = find( sum(blah).'~=0 );
```

D.3.24 SensFEMupdating

Detta är en implementering av "Sensitivity method finite element updating", som beskriven i [MLF11]. Den används ej för närvarande (och har ej testkörts med senaste versionen av övrig källkod i detta appendix), men kan, som besskrivs i Avsnitt 4.1.1 användas som ett alternativ till optimiseringsmetoden "Newton trust region" om denna för någon konstruktion ger konvergensproblem på grund av problem att räkna ut bra uppskattningar av andraderivator av målfunktionen. Ungefär som i Avsnitt D.2 sätter vi

$$\boldsymbol{z}^{\mathrm{FEM}}(\boldsymbol{a}) \stackrel{\mathrm{def}}{=} \begin{bmatrix} (2\pi\nu_1^{\mathrm{FEM}}(\boldsymbol{a}))^2 \\ \vdots \\ (2\pi\nu_M^{\mathrm{FEM}}(\boldsymbol{a}))^2 \\ \frac{\phi_{1,1,1}^{\mathrm{FEM}}(\boldsymbol{a})}{\phi_{1,1,n_{\mathrm{ref}}}^{\mathrm{FEM}}(\boldsymbol{a})} \\ \vdots \\ \frac{\phi_{D,M,N}^{\mathrm{FEM}}(\boldsymbol{a})}{\phi_{D,M,n_{\mathrm{ref}}}^{\mathrm{FEM}}(\boldsymbol{a})} \end{bmatrix} \quad \text{and} \quad \boldsymbol{z}^{\mathrm{mea}} \stackrel{\mathrm{def}}{=} \begin{bmatrix} (2\pi\nu_1^{\mathrm{mea}})^2 \\ \vdots \\ (2\pi\nu_M^{\mathrm{mea}})^2 \\ \frac{\phi_{1,1,1}^{\mathrm{mea}}}{\phi_{1,1,n_{\mathrm{ref}}}^{\mathrm{mea}}} \\ \vdots \\ \frac{\phi_{D,M,N}^{\mathrm{mea}}}{\phi_{D,M,n_{\mathrm{ref}}}^{\mathrm{mea}}} \end{bmatrix}$$

med någon omordning $\rho(d,m,n)$ av index (d,m,n) till heltalen $1,2,\ldots,DMN$. Algoritmen är iterativ, och i iteration nummer ith betraktar vi felet

$$\begin{aligned} \varepsilon_{\boldsymbol{z}^{\mathrm{FEM}}(\boldsymbol{a})} &\stackrel{\mathrm{def}}{=} \boldsymbol{z}^{\mathrm{mea}} - \boldsymbol{z}^{\mathrm{FEM}}(\boldsymbol{a}) = \boldsymbol{z}^{\mathrm{mea}} - \boldsymbol{z}^{\mathrm{FEM}}(\boldsymbol{a}_i) + \boldsymbol{z}^{\mathrm{FEM}}(\boldsymbol{a}_i) - \boldsymbol{z}^{\mathrm{FEM}}(\boldsymbol{a}) \\ &\stackrel{\mathrm{def}}{=} \boldsymbol{r}_i + \boldsymbol{z}^{\mathrm{FEM}}(\boldsymbol{a}_i) - \boldsymbol{z}^{\mathrm{FEM}}(\boldsymbol{a}) \approx \boldsymbol{r}_i - G_i(\boldsymbol{a} - \boldsymbol{a}_i) \\ &\stackrel{\mathrm{def}}{=} \boldsymbol{r}_i - G_i \boldsymbol{\Delta a}_i, \qquad (G_i)_{\rho,e} \stackrel{\mathrm{def}}{=} \frac{\partial (\boldsymbol{z}^{\mathrm{FEM}}(\boldsymbol{a}))_\rho}{\partial a_e}. \end{aligned}$$

Vi vill minimera $J(\boldsymbol{a}) \stackrel{\mathrm{def}}{=} \varepsilon_{\boldsymbol{z}^{\mathrm{FEM}}(\boldsymbol{a})}^{\mathrm{T}} W_\varepsilon \varepsilon_{\boldsymbol{z}^{\mathrm{FEM}}(\boldsymbol{a})}$, där det föreslås i [MLF11] att välja vikter

$$W_\varepsilon = [\mathrm{diag}(\boldsymbol{z}^{\mathrm{mea}})]^{-2},$$

där `diag` konstruerar en diagonalmatris av ev vektor, precis som MATLAB-kommandot med samma namn. Huvudalgoritmen i [MLF11] för att minimera $J(\boldsymbol{a})$ gör först en en minstakvadratlösning av $\boldsymbol{r}_i - G_i \boldsymbol{\Delta a}_i = \boldsymbol{0}$ och sätter sedan $\boldsymbol{a}_{i+1} = \boldsymbol{a}_i + \boldsymbol{\Delta a}_i$. Närmare bestämt blir detta som följer:

$$\begin{aligned} G_i \boldsymbol{\Delta a}_i &= \boldsymbol{r}_i \\ G_i^{\mathrm{T}} W_\varepsilon G_i \boldsymbol{\Delta a}_i &= G_i^{\mathrm{T}} W_\varepsilon \boldsymbol{r}_i \qquad \text{(antag att } G_i^{\mathrm{T}} W_\varepsilon G_i \text{ är inverterbar)} \\ \boldsymbol{\Delta a}_i &= (G_i^{\mathrm{T}} W_\varepsilon G_i)^{-1} G_i^{\mathrm{T}} W_\varepsilon \boldsymbol{r}_i \qquad (= (W_\varepsilon^{1/2} G_i) \setminus W_\varepsilon^{1/2} \boldsymbol{r}_i \text{ i MATLAB-notation}). \end{aligned}$$

För att räkna ut derivatorna i G_i, så räknas derivatorna $\frac{\partial \phi_{d,m,n}^{\mathrm{FEM}}}{\partial a_e}$ och $\frac{\partial (2\pi\nu_m^{\mathrm{mea}})^2}{\partial a_e}$ ut med kommandot `FoxKapoor`, och kombineras (som i [TMDR02, TDR03, RTDR10] med sambandet

$$\frac{\partial \frac{\phi_{d,m,n}^{\mathrm{FEM}}(\boldsymbol{a})}{\phi_{d,m,n_{\mathrm{ref}}}^{\mathrm{FEM}}(\boldsymbol{a})}}{\partial a_e} = \frac{\frac{\partial \phi_{d,m,n}^{\mathrm{FEM}}}{\partial a_e} \phi_{d,m,n_{\mathrm{ref}}}^{\mathrm{FEM}} - \phi_{d,m,n}^{\mathrm{FEM}} \frac{\partial \phi_{d,m,n_{\mathrm{ref}}}^{\mathrm{FEM}}}{\partial a_e}}{{\phi_{d,m,n_{\mathrm{ref}}}^{\mathrm{FEM}}}^2}, \quad m = 1,\ldots,M, \quad n = 1,\ldots,N.$$

För illa konditionerade system utvidgades i [MLF11] algoritmen ovan genom att undersöka det regulariserade problemet att söka ett minimum för

$$J(\boldsymbol{a}) \stackrel{\mathrm{def}}{=} \varepsilon_{\boldsymbol{z}^{\mathrm{FEM}}(\boldsymbol{a})}^{\mathrm{T}} W_\varepsilon \varepsilon_{\boldsymbol{z}^{\mathrm{FEM}}(\boldsymbol{a})} + \lambda^2 \boldsymbol{\Delta a}_i^{\mathrm{T}} W_{\boldsymbol{a}} \boldsymbol{\Delta a}_i,$$

with the suggestion to choose

$$W_{\boldsymbol{a}} \stackrel{\text{def}}{=} \frac{\text{mean}(\text{diag}(\Gamma))}{\text{mean}(\text{diag}(\Gamma^{-1}))}\Gamma^{-1}, \qquad \Gamma \stackrel{\text{def}}{=} G_i^{\mathrm{T}} W_\varepsilon G_i.$$

I varje iterativt steg av algoritmen ges då $\boldsymbol{\Delta a}_i$ av formeln

$$\boldsymbol{\Delta a}_i = (G_i^{\mathrm{T}} W_\varepsilon G_i + \lambda^2 W_{\boldsymbol{a}})^{-1} G_i^{\mathrm{T}} W_\varepsilon \boldsymbol{r}_i.$$

I [MLF11] så föreslås att välja regulariseringsparametern λ^2 mellan 0 (ingen regularisering) och 0.3 (för många okänsliga parametrar och mycket illa konditionerad matris $G_i^{\mathrm{T}} W_\varepsilon G_i$, eller $\lambda^2 = 0.05$ för ett mindre illa kontitionerat fall.

```
function a = SensFEMupdating(lambda,a0,nRef,nuOMA,phiOMA, ...
                        NN,structAbaqusOriginalInput,abaqus_input_file,...
                        structBeam,masterNodes,masterModeNumbers,...
                        masterDOFs,RunWithAbaqus,RunWithModeReduction)
% Sensitivity method finite element updating as described in [1]
%
% USAGE:
%
% INPUT
% lambda = regularisation parameter. Suggested in [1] to choose lambda^2
%          between 0 (no regilarisation) and 0.3 (for many insensitive
%          parameters and strongly ill-condiitoned matrix
%          Gi'*Wepsilon*Gi, or lambda^2=0.05 if the ill-conditioning
%          is not too strong.
% a0     = a column vector containing the initial guess of the values
%          of the parameters a.
% nRef   =  used by tthe function ModeShapeResid for mode shape
%           normalization in the function ModeShapeResid if nRef is
%           nonzero. Otherwise ModeShapeResid chooses a suitable nRef.
% nuOMA  = lengt m vector containing the measured mode frequencies
% phiOMA = NxMxD array containing the measuren mode shapes as column
%          vectors
% NN                        = Input parameter for ComputeNewModes.
%                             Described there.
% structAbaqusOriginalInput = Input parameter for ComputeNewModes.
%                             Described there.
% abaqus_input_file         = Input parameter for ComputeNewModes.
%                             Described there.
% masterNodes               = Input parameter for ComputeNewModes.
%                             Described there.
% masterModeNumbers         = Input parameter for ComputeNewModes.
%                             Described there.
% masterDOFs                = Input parameter for ComputeNewModes.
%                             Described there.
%
% OUTPUT
%
% [1] John E. Mottershead, Michael Link and Michael I. Friswell,
%     The sensitivity method in finite element model updating: A tutorial.
%     Mechanical Systems and Signal Processing 25, 2011, 2275-2296.

global phiFEM % NxMxD array with node points of the mth mode in the columns
global nuFEM  % Length M column vector with the frequency of each mode
global K_0e   % NxNxE array of NxN element stiffness matrices

% Pick one:
%ShapeResType='diff';
 ShapeResType='l_2';  % Shouldn't this be chosen globally in FemUpdating.m?????????

MaxNOFiter = 100;
DeltaaiThres = 100*eps; % Threshold for ending iterations
[N,M,D] = size(phiFEM);
R = M*(D*N+1); % Number of rows in Gi, z and r
E=length(K_0e(1,1,:));
Gi=zeros(R,E);
```

```
Deltaai=Inf;
IterNr=1;
a=a0;
while ( (Deltaai >= DeltaaiThres) && ( IterNr<= MaxNOFiter ) )

  r_f = (nuFEM.^2 - nuOMA.^2); % ./nuOMA.^2; <-- Now in W
  [r_s,nRef,phiRelOMA,phiRelFEM] ...
                          = ModeShapeResid(phiFEM,phiOMA,ShapeResType,nRef);
  %[r,w]=ResidAndWeights(r_f,r_s,WeightingStrategy);

  r=[r_f(:) ; r_s(:)]; % Tre residual



  zFEM = [ (2*pi*nuFEM(:)).^2
           phiRelFEM(:) ];
  zOMA = [ (2*pi*nuOMA(:)).^2
           phiRelOMA(:) ];
  WepsilonDiag=zOMA;
  ind = find(abs(zOMA)<100*eps);
  WepsilonDiag(ind)=1; % No normalization in node points (to avoid div with 0)
  if strcmp(ShapeResType,'l_2')
    WepsilonDiag(M+1:end)=1;
  end
  Wepsilon = diag(WepsilonDiag.^(-2)); % Weighting matrix suggested in [1, p. 2279]
  sqrtWepsilon=Wepsilon.^(1/2);

  %% ==== Compute Gi matrix elements ================================
  % Compute the Fox-Kapoor formulas (12) and (13) in "Damage assessment..."
  [d_omegam2_d_ae,d_phim_d_ae] = FoxKapoor(phiFEM,nuFEM,K_0e);

  % Since the each mode shapes in z is normalized so that element nr nRef
  % equals 1, we need to compute the derivative of a quotient:
  dz_rho_da_e = zeros(N,M,D,E); % Derivative of the quotient
  for dd=1:D
   phi_mnRefDiag=diag(phiFEM(nRef,:,dd));             % The same factor for all e
   phi_mnRefDiagPowMin2=diag(phiFEM(nRef,:,dd).^(-2));% and the same denominator
    for ee=1:E
      if strcmp(ShapeResType,'diff')
        dz_rho_da_e(:,:,dd,ee) = ( d_phim_d_ae(:,:,dd,ee)*phi_mnRefDiag ...
                                   - phiFEM(:,:,dd)*diag(d_phim_d_ae(nRef,:,dd,ee)) ...
                                 ) ...
                                 *phi_mnRefDiagPowMin2;
      elseif strcmp(ShapeResType,'l_2')
        for mm=1:M
          normPHIfem = sqrt( phiFEM(:,mm,dd).'*phiFEM(:,mm,dd) );
          dz_rho_da_e(:,mm,dd,ee) = d_phim_d_ae(:,mm,dd,ee)./normPHIfem ...
                                    - d_phim_d_ae(:,mm,dd,ee).'*phiFEM(:,mm,dd) ...
                                      *phiFEM(:,mm,dd)./ (normPHIfem^3);
        end
      else
        error('Not yet implemented.')
      end
    end
  end
  Gi = [ d_omegam2_d_ae  % with omegam2 = (2*pi*nu_m).^2. MxE-matrix
         reshape(dz_rho_da_e,N*M*D,E) ];
%    Ji=Jacobian(phiFEM,nuFEM,'diff',nRef,phiOMA,nuOMA,K_0e)

  if ( lambda == 0 )
    %% ==== Least square solution ====================================
    Deltaai = (sqrtWepsilon*Gi)\(sqrtWepsilon*r)
  else
    %Gammas = diag(Gi.'*Wepsilon*Gi);  % Longer name since Gamma is occupied
    Gammas =      (Gi.'*Wepsilon*Gi);  % Longer name since Gamma is occupied
    invGammas=inv(Gammas);
    Wa      = mean(diag(Gammas)) ...
              / mean(diag(invGammas)) ...
              *invGammas;  % Guessing how to interpret the typo in [1]...???
    Deltaai = inv(Gi.'*Wepsilon*Gi +lambda^2*Wa)*Gi.'*Wepsilon*r;
  end
  a=a+Deltaai;
  ComputeNewModes(a,[],[],NN,structAbaqusOriginalInput,abaqus_input_file,...
```

```
                    structBeam,masterNodes,masterModeNumbers,masterDOFs,...
                    RunWithAbaqus,RunWithModeReduction);
  IterNr=IterNr+1
end

if (IterNr == MaxNOFiter )
  error(['The maximun number of iterations (' num2str(MaxNOFiter) ...
         ') was npot enough for convergence'])
end
```

D.3.25 UpdateAbaqusInpFile.m

Anropas av `ComputeNewModes.m` och `InitAbaqusModelGroupsOLD.m` för att spara nya värden på uppdateringsparanmetrar till en av Abaqus indatafiler.

```
function UpdateAbaqusInpFile(AbaqusInpFilName,AbaqusModelPartFilName,...
                             AbaqusHistPartFilName,StructMaterial)

copyfile(AbaqusModelPartFilName,AbaqusInpFilName);
MaterialSection = BuildAbaqusMaterialSection(StructMaterial);
fid   = fopen(AbaqusInpFilName,'at+'); % open for reading and writing
fprintf(fid,'%s',MaterialSection);
fclose(fid);

fid   = fopen(AbaqusInpFilName,'a'); % open for readng and writing
fida  = fopen(AbaqusHistPartFilName,'r');

tline = fgets(fida);

while ischar(tline)
    fprintf(fid,'%s',tline);
    tline = fgets(fida);
end;

% CHANGE to system('type f1 f2 > f3')
fclose(fid);
fclose(fida);

function str = BuildAbaqusMaterialSection(StructMaterial)

materials     = '** MATERIALS';
stars         = '** ';
material_name = '*Material, name=';
density       = '*Density';
elastic       = '*Elastic';

n   = length(StructMaterial);

str = sprintf('%s\n%s\n%s\n',stars,materials,stars);

for index=1:n
    str = sprintf('%s\n',[str material_name StructMaterial(index).Name]);
    str = sprintf('%s\n',[str density]);
    str = sprintf('%s%f,\n',str,StructMaterial(index).Density);
    str = sprintf('%s\n',[str elastic]);
    str = sprintf('%s %g, %f\n',str,StructMaterial(index).Elastic, ...
        StructMaterial(index).Poisson);
end

str = sprintf('%s\n',[str stars]);
```

D.3.26 Weights.m

Anropas av `InitAbaqusModelGroupsOLD.m` för att sätta värde på `OptConst.Weights`.

```
function w=Weights(r_fLen,r_sLen,WeightingStrategy)
% USAGE: [r,w]=ResidAndWeights(r_f,r_s,WeightingStrategy)
%
% Combines the frequency residuals r_f and the shape residuals r_s to a
% column vector r and computes corresponding weights for weighted least
% squares measures of the size of r.
%
% INPUT
% r_f
% r_s
% WeightingStrategy
%
% OUTPUT
% r = column vector r=[r_f(:) ; r_s(:)]
% w = weights (column vector of the same length as r)
if ( nargin<3 )
  WeightingStrategy = 'none'
end

%r_f = r_f(:); % Make column vector
%r_s = r_s(:); % Make column vector
%r_fLen = length(r_f);
%r_sLen = length(r_s);

if strcmp(WeightingStrategy,'none')
  w = [ ones(r_fLen,1)
        ones(r_sLen,1) ];
elseif strcmp(WeightingStrategy,'test')
  %w = [ ones(r_fLen,1)
  %      ones(r_sLen,1) ];

      w = [ zeros(r_fLen,1)
        zeros(r_sLen,1) ];

    w(1:7) = 1;
    dofsPerMode=r_sLen/r_fLen;
    % mode no.3 : substract [2,3]
    w(end-(r_fLen)*dofsPerMode+1:end-(r_fLen-1)*dofsPerMode)=1; % 1
     w(end-(r_fLen-1)*dofsPerMode+1:end-(r_fLen-2)*dofsPerMode)=1; % 2
     w(end-(r_fLen-2)*dofsPerMode+1:end-(r_fLen-3)*dofsPerMode)=1; % 3
    w(end-(r_fLen-3)*dofsPerMode+1:end-(r_fLen-4)*dofsPerMode)=1; % 4
     w(end-(r_fLen-4)*dofsPerMode+1:end-(r_fLen-5)*dofsPerMode)=1; % 5
     w(end-(r_fLen-5)*dofsPerMode+1:end-(r_fLen-6)*dofsPerMode)=1; % 6
     w(end-(r_fLen-6)*dofsPerMode+1:end-(r_fLen-7)*dofsPerMode)=1; % 7
elseif strcmp(WeightingStrategy,'averaging')
  w = [ ones(r_fLen,1)./r_fLen
        ones(r_sLen,1)./r_sLen ];
elseif strcmp(WeightingStrategy,'mod averaging')
  % Modified averaging, with the weight for mode shapes mad 1000 times
  % smaller which for one testing setup for the plate was roughly what's
  % needed for putting almost the same average weight in the mode shapes as
  % the weighting of each frequency in the objective function.
  % (If you always want the same weight on every frequency and every mode
  % shape for any experimental setup, consider no weigting and l2
  % norlalizatio instead).
  w = [ ones(r_fLen,1)./r_fLen
        ones(r_sLen,1)./r_sLen./100];
  warning('Temporary change to dividing')
elseif strcmp(WeightingStrategy,'FreqOnly')
% elseif strcmp(WeightingStrategy,'norm1')
%   w = [ ones(r_fLen,1)./norm(r_f)^2
%         ones(r_sLen,1)./norm(r_s)^2 ];
  w = [ ones(r_fLen,1)
        zeros(r_sLen,1) ];
else
  error('Incorrect input WeightingStrategy')
end
```

D.3.27 zCoordsForPlane.m

Anropas av `InterpolFunctions.m`.

```
function z = zForPlane(PointsInPlane,x,y)
% For a specified plane P, this function computes the z-coordinates for
% points with x- and y-coordinates
%
% INPUTS
% PointsInPlane = 3x3 matrix with column vectors giving the location for
%                 three points in the plane P.
% x,y           = numbers or arrays of the same size.
%
% OUTPUTS
% z = number or array the same size as x and y, with z(m,n,...) being the
%     z-coordinate for the point on the plane with x- and y-coordinates
%     x(m,n,...) and y(m,n,...), respevtively.

%% ==== Compute normal vector for the plane ================================
a=PointsInPlane(:,1);
b=PointsInPlane(:,2);
c=PointsInPlane(:,3);
n = cross(b-a,c-a);
if n(3)==0
  error('Plane parallel with z-axis. No uniqely determined z-coordinate.')
end

%% ==== Standard formula for points in the plane ===========================
z=-( (x-a(1))*n(1) + (y-a(2))*n(2) )./n(3) + a(3);
```

Referenser

[Bri13] Rune Brincker. *Book manuscript, in preparation*. 2013.

[BVA01] Rune Brincker, Carlos E. Ventura, and Palle Andersen. Damping estimation by frequency domain decomposition. In *Proceedings of the 21st International Modal Analysis Conference on Structural Dynamics (IMAC)*, pages 698–703, Kissimmee, Fla, USA, February 2001. WWW: http://www.svibs.com/solutions/literature/2001_4.pdf.

[BVA03] Rune Brincker, Carlos E. Ventura, and Palle Andersen. Why output-only modal testing is a desirable tool for a wide range of practical applications. In *Proceedings of the 21st International Modal Analysis Conference on Structural Dynamics (IMAC)*, pages 1–8, Kissimmee, Fla, USA, February 2003. WWW: `ftp://ftp.svibs.com/Download/Literature/Papers/2003/2003_3.pdf`.

[BZA01] Rune Brincker, Lingmi Zhang, and Palle Andersen. Modal identification of output-only systems using frequency domain decomposition. *Smart Mater. Struct.*, 10(3):441–445, June 2001. DOI: 10.1088/0964-1726/10/3/303.

[CJK06] Roy R. Craig Jr. and Andrew J. Kurdila. *Fundamentals of structural dynamics*. Wiley, Hoboken, NJ, second edition, 2006.

[CLL09] Shyh-Leh Chen, Jia-Jung Liua, and Hsiu-Chi Laia. Wavelet analysis for identification of damping ratios and natural frequencies. *J. Sound Vib.*, 323(1–2):130–147, June 2009. DOI: http://10.1016/j.jsv.2009.01.029.

[EGOS09] Lennart Elfgren, Niklas Grip, Ulf Ohlsson, and Natalia Sabourova. Signal processing in modal analysis of bridges. Slides and notes from a talk given at the workshop "Assessment of Bridges in China and Sweden", Luleå University of Technology, December 21st 2009, December 2009. WWW: http://pure.ltu.se/portal/files/5135812/Signal_processing_in_modal_analysis_of_bridges._Slides_and_notes.

[FGS12] Thomas Forsberg, Niklas Grip, and Natalia Sabourova. Non-iterative calibration for accelerometers with three non-orthogonal axes, reliable measurement setups and simple supplementary equipment. *Meas. Sci. Technol.*, submitted:20 pages, 2012. DOI: 10.1088/0957-0233/24/3/035002, WWW: http://pure.ltu.se/portal/files/41270184/FGS12Preprint.pdf.

[FK68] R. L. Fox and R. M. Kapoor. Rate of change of eigenvalues and eigenvectors. *AIAA J.*, 6(12):2426–2429, December 1968.

[FM95] M.I. Friswell and J.E. Mottershead. *Finite Element Model Updating in Structural Dynamics*. Kluwer Academic, Dordrecht, 1995.

[GGN13] Niklas Grip, Carl-Erik Grip, and Leif Nilsson. Wavelet study of dynamic variations in steel and ironmaking rest gases. Potential effect on external use. *Applied Energy*, 112:1032–1040, December 2013. DOI: 10.1016/j.apenergy.2013.05.034, WWW: http://pure.ltu.se/portal/files/42043630/RevisedPreprint.pdf.

[GS11] Niklas Grip and Natalia Sabourova. Simple non-iterative calibration for triaxial accelerometers. *Meas. Sci. Technol.*, 22(12):13 pages, December 2011.

WWW: http://pure.ltu.se/portal/files/32843182/GrSa11a.pdf, DOI: 10.1088/0957-0233/22/12/125103.

[HZE+08] G. He, Z. Zou, O. Enochsson, A. Bennitz, L. Elfgren, A. Kronborg, B. Töyrä, and B. Paulsson. Assessment of railway concrete arch bridge by numerical modelling and measurements. In H.-M. Koh and D. M. Frangopol, editors, *Bridge Maintenance, Safety, Management, Health Monitoring and Informatics*, pages Abstract p 722 + full version on CD pp 3733–3742, Leiden, 2008. CRC Press/Balkema, Taylor & Francis.

[Jai05] Bijaya Jaishi. *FINITE ELEMENT MODEL UPDATING OF CIVIL ENGINEERING STRUCTURES UNDER OPERATIONAL CONDITIONS*. Doctoral thesis, College of Civil Engineering and Architecture, Fuzhou University, Fujian, China, May 2005. WWW: http://bridge.csu.edu.cn/english/FileAccup/200505151713.pdf.

[Mai11] Maintenance, renewal and improvement of rail transport infrastructure to reduce economic and environmental impacts (mainline), 2011. A European Community 7th Framework Program research project 2011–2014 with 19 partners. Grant agreement 285121, SST.2011.5.2-6. Dr. Björn Paulsson, UIC/Trafikverket acts as Project Coordinator, see http://www.mainline-project.eu.

[Mar10] Tshilidzi Marwala. *Finite Element Model Updating Using Computational Intelligence Techniques: Applications to Structural Dynamics*. Kluwer Academic, 2010.

[Mat10] The MathWorks Inc., Natick, MA, USA. *Optimization Toolbox™ 5 User's Guide*, September 2010. WWW: http://???

[MLF11] John E. Mottershead, Michael Link, and Michael I. Friswell. The sensitivity method in finite element model updating: A tutorial. *Mech. Sys. Sig. Proc.*, 25(7):2275–2296, October 2011. DOI: 10.1016/j.ymssp.2010.10.012.

[NW06] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer Series in Operations Research. Second edition, 2006. WWW: http://faculty.bracu.ac.bd/∼faruqe/books/numericalanalysis/Numerical%20Optimization%202ed.pdf.

[OM96] Peter van Overschee and Bart de Moor. *Subspace Identification for Linear Systems: Theory—Implementation—Applications*. Kluwer, 1996. WWW: http://www.shadmehrlab.org/Courses/learningtheory_files/subspace_book_96.pdf.

[PDR99] Bart Peeters and Guido De Roeck. Reference-based stochastic subspace identification for output-only modal analysis. *Mech. Sys. Sig. Proc.*, 13(6):855–878, November 1999. DOI: 10.1006/mssp.1999.1249.

[PDR01] Bart Peeters and Guido De Roeck. Stochastic system identification for operational modal analysis: A review. *J. Dyn. Sys., Meas., Control*, 123(4):659–667, December 2001. DOI: 10.1115/1.1410370.

[RDR08] Edwin Reynders and Guido De Roeck. Reference-based combined deterministic–stochastic subspace identification for experimental and operational modal analysis. *Mech. Sys. Sig. Proc.*, 22(3):617–637, April 2008. DOI: 10.1016/j.ymssp.2007.09.004.

[Ric97] Mark H. Richardson. Is it a mode shape, or an operating deflection shape? *Sound & Vibration Magazine* 30th *Anniversary Issue*, March 1997. WWW: http://www.systemplus.co.jp/support/data/techpaper/mescope/tech/10.pdf.

[RPDR08] Edwin Reynders, Rik Pintelon, and Guido De Roeck. Uncertainty bounds on modal parameters obtained from stochastic subspace identification. *Mech. Sys. Sig. Proc.*, 22(4):948–969, May 2008. DOI: 10.1016/j.ymssp.2007.10.009.

[RTDR10] Edwin Reynders, Anne Teughels, and Guido De Roeck. Finite element model updating and structural damage identification using OMAX data. *Mech. Sys. Sig. Proc.*, 24(5):1306–1323, July 2010. DOI: 10.1016/j.ymssp.2010.03.014.

[Sab11] Natalia Sabourova. Dynamic response of the Kirunaviadukten bridge. 24 page appendix of the research report *Gruvvägsbron i Kiruna - Översyn av deformationskapacitet* by Ola Enochsson, Natalia Sabourova, Lennart Elfgren and Mats Emborg, Department of Civil and Environmental Engineering, Division of Structural Engineering, Luleå University of Technology, to appear 2011.

[SD+09] Wei Song, , Shirley Dyke, GunJin Yun, and Thomas Harmon. Improved damage localization and quantification using subset selection. *J Eng Mech-ASCE*, 135(6):548–560, June 2009. DOI: 10.1061/(ASCE)EM.1943-7889.0000005.

[SGP+12] Natalia Sabourova, Niklas Grip, Arto Puurula, Ola Enochsson, Yongming Tu, Ulf Ohlsson, Martin Nilsson, Lennart Elfgren, Anders Carolin, and Håkan Thun. The railway concrete arch bridge over Kalix River — Dynamic properties and load carrying capacity. In Dirch H Bager and Johan Silfwerbrand, editors, *Proceedings of fib Symposium: Concrete Structures for Sustainable Community*, pages 609–612, Royal Institute of Technology (KTH), Stockholm, Sweden, June 2012. WWW: http://www.fibstockholm2012.se.

[SPG09] Hendrik Schlune, Mario Plos, and Kent Gylltoft. Improved bridge evaluation through finite element model updating using static and dynamic measurements. *Eng. Struct.*, 31(7):14771485, July 2009. DOI: 10.1016/j.engstruct.2009.02.011.

[Sus08] Sustainable bridges – assessment for future traffic demands and longer lives, 2008. A European Integrated Research Project during 2003–2008. Four guidelines and 35 background documents are available at http://www.sustainablebridges.net.

[Sva10] P-O Svahn. Kv. klassföreståndaren, stockholm störande bjälklagsvibrationer — sammanfattande rapport av undersökningar, analyser och åtgärder. Teknisk rapport, Skanska Sverige AB, Teknik och Projekteringsledning — Bro och Anläggning, Göteborg, June 2010. Collection of articles that were originally published in Experimental Techniques. WWW: http://sdasl.uml.edu/umlspace/mspace.html.

[TDR03] Anne Teughels and Guido De Roeck. Damage assessment of the Z24 bridge by FE model updating. *Key Eng. Mat.*, 245–246:19–26, July 2003. DOI: 10.4028/www.scientific.net/KEM.245-246.19.

[TMDR02] Anne Teughels, Johan Maeck, and Guido De Roeck. Damage assessment by FE model updating using damage functions. *Comput Struct*, 80(25):1869–1879, September 2002. DOI: 10.1016/S0045-7949(02)00217-1.