ESTRATÉGIAS ADAPTATIVAS MULTINÍVEL PARA A SOLUÇÃO DE PROBLEMAS NÃO-LINEARES PELO MÉTODO DOS ELEMENTOS FINITOS.

Claudio Ernesto Jouglard

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO EM ENGENHARIA DA UNIVERSIDADE FEDERAL DE RIO DE JANEIRO COMO PARTE DOS REQUISITOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA CIVIL.

Aprovada por:

Prof. Ronaldo Carvalho Batista, Ph. D.
(Presidente)

Prof. Alvaro Luiz Gayoso de Azeredo Coutinho, D. Sc.

Prof. José Luis Drummond Alves, D. Sc.

Prof. Abimael Fernando Dourado Loula, D. Sc.

Prof. Luiz Fernando Campos Ramos Martha, Ph. D.

RIO DE JANEIRO, RJ - BRASIL
JUNHO DE 1996

JOUGLARD, CLAUDIO ERNESTO

Estratégias Adaptativas Multinível para a Solução de Problemas Não-lineares pelo Método dos Elementos Finitos (Rio de Janeiro) 1996.
VII, 34 p., 29,7 cm (COPPE/UFRJ, D.Sc., Engenharia Civil, 1996)
Tese - Universidade Federal do Rio de Janeiro, COPPE.
1. Análise adaptativa com elementos finitos.
I. COPPE/UFRJ II Título (série)

A mi mujer, Liliana, por la infinita paciencia y estimulos recibidos.

AGRADECIMENTOS

Ao Prof. Ronaldo Carvalho Batista pela orientação, incentivo e amizade dispensadas durante a execução deste trabalho.

Ao Prof. Alvaro Coutinho pela orientação e pelo interesse dispensado a este trabalho.

Ao CNPq pelo apoio financeiro na forma de bolsa de estudos.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.).

ESTRATÉGIAS ADAPTATIVAS MULTINÍVEL PARA A SOLUÇÃO DE PROBLEMAS NÃO-LINEARES PELO MÉTODO DOS ELEMENTOS FINITOS.

Claudio Ernesto Jouglard

JUNHO, 1996

Orientadores:Prof. Ronaldo Carvalho Batista e Prof. Alvaro Luiz Gayoso de Azeredo Coutinho.

Programa : Engenharia Civil.

Quando é empregado um método numérico aproximado, como o método dos elementos finitos, para solucionar um modelo matemático de um problema fisico são introduzidos erros de discretização devidos à aproximação do modelo matemático contínuo por um modelo discreto. Uma discretização é caracterizada pela malha de elementos finitos e o tipo de elemento. O objetivo de uma análise adaptativa é controlar o erro de discretização incrementando o número de graus de liberdade nas regiões da malha onde o modelo prévio de elementos finitos não é adequado e atingir uma certa precisão da maneira mais económica possível. Um método adaptativo deve ter dois componentes principais: um estimador de erro e uma estratégia de otimização da malha.

Neste trabalho se apresentam estratégias adaptativas para problemas de elasticidade bidimensional. É apresentado um estimador de erro baseado no promedio dos fluxos nas interfaces entre elementos. São usadas coordenadas locais para definir os fluxos normais e tangenciais em cada lado de elemento. Isto permite considerar as condições de contorno sobre os fluxos de uma maneira simple e natural no estimador de erro, o qual fica muito mais robusto e confiavél.

Também é apresentada uma estrutura de dados para o código adaptativo, a qual é organizada por níveis, e consiste na estrutura de dados tradicional do método de elementos finitos suplementada por arranjos adicionais com informação de conetividades.

É feita uma comparação do desempenho de solucionadores multinível que tiram proveito do ordenamento hierárquico das diferentes malhas de elementos finitos geradas adaptativamente. Destas comparações o método dos gradientes conjugados com precondicionador hierarquico multinível resulta o mais eficiente.

A estratégia adaptativa e o solucionador multinível foram combinados para a solução de problemas de elasticidade bidimensional com não linearidade geométrica. Dos resultados obtidos pode-se concluir que a metodologia adotada é adequada para solucionar este tipo de problemas.

Abstract of Thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.).

A MULTILEVEL ADAPTIVE APPROACH FOR THE SOLUTION OF NON-LINEAR FINITE ELEMENT PROBLEMS.

Claudio Ernesto Jouglard

JUNE, 1996

Thesis Supervisors:Prof. Ronaldo Carvalho Batista and Prof. Alvaro Luiz Gayoso de Azeredo Coutinho.

Department : Civil Engineering.

When using an approximate numerical method, such as the finite element method, to solve a mathematical model of a physical problem, discretization errors are introduzed due to the approximation of the continuous mathematical model by a discrete model. A discretization is characterized by the finite element mesh and the choice of elements. The objective of an adaptive analysis is to control the discretization error by increasing the number of degrees of freedom in regions where the previous finite element model is not adequate and to achieve the specified accuracy in a most economic manner. An adaptive method must have two major components: an error estimator and a mesh optimization strategy.

In this work we present an adaptive approach to two-dimensional elasticity problems. Also we present an error estimator based on the averaged fluxes at element interfaces. We use local coordinates to define the normal and tangential fluxes on each side of the element. Then we can consider the prescribed fluxes in the error estimator in a simple and natural manner, resulting in robust and reliable error estimates.

Also, we present a data structure for the adaptive code, which is organized by levels, and consists of the traditional finite element data structure supplemented by additional arrays of connectivity information.

We made a comparison of multilevel solvers which are particularly suited for the hierarchical ordering of adaptively generated finite element meshes. From these comparisons the conjugate gradients method with multilevel hierarchical preconditioner was the most efficient.

The adaptive strategy and the multilever solver were combined for the solution of two-dimensional elasticity problems with geometric nonlinearities. From the obtained results we can conclude that the methodology adopted is adequate to solve this type of problems.

Sumário

Capítulo 1

Descrição Geral

Adaptatividade é um termo introduzido no início da década de 80 para descrever um processo automático de convergência que procura distribuir de forma ótima os paramêtros básicos de aproximação do método dos elementos finitos (MEF). Dentro do MEF as estratégias adaptativas mais frequentemente utilizadas são classificadas como refinamentos h e p. No primeiro caso, a precisão desejada é alcançada diminuindo-se o tamanho dos elementos, enquanto que no outro caso, aumenta-se a ordem p dos polinômios que definem as funções de forma.

Está implícito nos métodos adaptativos a habilidade de acessar, de alguma forma, a "qualidade" da aproximação local sobre vários elementos em uma malha. Esta qualidade da aproximação numérica é julgada naturalmente pelo erro associado, isto é, a diferença entre a solução exata u e a aproximação de elementos finitos, u_h, medida em uma norma associada. Portanto, uma parte integrante de qualquer esquema racional para adaptatividade é algum meio de estimar, geralmente *a-posteriori*, o erro de aproximação local. As estimativas *a-posteriori* são obtidas à partir de uma solução pré-existente.

Em Jouglard [1], apresenta-se um estimador de erro para triângulos lineares baseado em projeção de tensões. Tal estimador emprega coordenadas locais para definir fluxos normais e tangenciais em cada lado dos elementos. Com estes valores calculam-se os fluxos normais e tangenciais médios em cada lado, empregando-se os valores dos triângulos adjacentes. Nos contornos onde existem carregamentos aplicados, o gradiente suavizado da solução pode ser forçado à adotar os valores ponderados dos fluxos prescritos. Propõem-se em [1] técnicas de ponderação dos fluxos quando há transição de lados ou cargas concentradas. Além disso, nas partes do contorno onde prescrevem-se deslocamentos é conveniente empregar-se um esquema melhorado para cálculo das tensões. O método desenvolvido em [1] consiste no cálculo de uma distribuição de tensões equivalente às reações nodais. Estas forças são obtidas por equilíbrio em todos os pontos nodais da malha, e redistribuídas usando o esquema anterior para cargas concentradas. Assim, as forças agindo em cada lado dos elementos são iguais, mesmo que o campo de tensões seja descontínuo devido à variações abruptas de material ou espessura. Deve-se notar que este tratamento para as

descontinuidades pode ser útil em outras áreas, como por exemplo, escoamentos em meios porosos, onde a descrição geológica do solo, gerada tipicamente através do uso combinado de dados medidos e geoestatísticos, produz grandes variações nas propriedades físicas.

As tensões preditas pelo método anterior são contínuas somente nos pontos médios dos lados. Esta falta de continuidade também serve como medida da qualidade da aproximação numérica, complementando as formas mais abstratas de medida, baseadas na norma de energia da solução.

Em [1] compara-se o desempenho do estimador de erro descrito acima , em problemas de elasticidade plana, com diversos outros descritos na literatura.Através da comparação de índices de efetividade e erros, conclui-se que o estimador proposto possui excelente desempenho, desde que sejam incluídos os fluxos externos, avaliados como descrito anteriormente.

Um dos pontos fundamentais para uma implementação adequada dos métodos adaptativos é a estrutura de dados empregada. Em [1] descreve-se uma estrutura de dados organizada por níveis de refinamento, que consiste basicamente da estrutura de dados convencional do método dos elementos finitos suplementada por arranjos adicionais, contendo informações sobre o inter-relacionamento dos elementos. Deve-se salientar que ao se projetar uma estrutura de dados, existe um compromisso entre a demanda de memória e o correspondente tempo de processamento. Este compromisso deve ser o mais balanceado possível. A estrutura de dados empregada em [1] contém informações sobre a posição dos elementos e lados na árvore de refinamento.

No contexto de análises adaptativas, os métodos iterativos de solução de sistemas de equações lineares são os mais adequados. Isto se deve basicamente ao fato de que se conhece uma boa aproximação da solução na malha corrente, geralmente obtida pela projeção da solução da malha anterior. Entretanto, o desempenho dos métodos iterativos é fortemente influenciado pelos valores numéricos dos dados do problema à ser resolvido. Em consequência, tem-se como objetivo o desenvolvimento de métodos iterativos robustos, que tenham um bom desempenho para uma gama variada de dados. Entre os métodos iterativos mais eficientes encontra-se o Método dos Gradientes Conjugados Precondicionado (MGCP). A eficiência do MGCP depende fortemente do tipo de precondicionador utilizado. Uma outra família de algoritmos iterativos muito eficientes são os Métodos Multi-malha, os quais requerem uma hierarquia de discretizações, isto é, empregam diversas malhas de densidade crescente na solução de um dado problema. O objetivo principal dos métodos multi-malha é se obter um método iterativo de solução onde a quantidade de trabalho computacional seja diretamente proporcional ao número de incógnitas do problema.

Se a base nodal padrão do MEF é substituída por bases hierárquicas quando uma malha de um dado nível é refinada, a discretização resultante preserva todas as informações dos níveis precedentes. Quando métodos multi-malha são utilizados em conjunto com bases hierárquicas, tem-se o Método Multi-malha com Base Hierárquica (MBH). A análise matemática do método MBH mostra que ele compartilha as propriedades básicas dos métodos multi-malha padrão.

Em [1] propõem-se precondicionadores multi-nível hierárquicos para o MGCP, baseados em uma aproximação hierárquica das matrizes de elementos finitos resultantes. Além disso, se desenvolve um critério de convergência para o MGCP baseado na Variação de energia entre iterações sucessivas do MGCP. Discute-se em [1] a utilização deste critério no contexto de adaptatividade e mostra-se que a escolha da tolerância para o MGCP deve se basear na qualidade do estimador de erro e não no número de graus de liberdade das malhas adaptadas. Note que o critério de energia proposto tem a vantagem adicional de ser adimensional.

Comparações numéricas foram efetuadas entre diversos solucionadores iterativos no contexto de adaptatividade em problemas lineares de elasticidade plana [1]. À partir dos resultados, verifica-se que o MGCP com precondicionador hierárquico se mostrou o mais eficiente. Os métodos multi-malha foram mais lentos e demandaram mais memória que a estratégia anterior. A superioridade do MGCP com precondicionador hierárquico sobre os solucionadores multi-malha em todos os exemplos testados é um fato importante, uma vez que os métodos multi-malha dependem de paramêtros que devem ser determinados empiricamente, através de experimentos numéricos. Tal nao ocorre com o MGCP com precondicionador hierárquico.

Analisa-se em [1] alguns exemplos de problemas não-lineares geométricos empregando-se a estratégia adaptativa e o MGCP com precondicionador hierárquico. Os resultados mostram que a estratégia é adequada para o tratamento de problemas desta natureza. Entretanto, deve-se notar que estruturas prismáticas, como vigas ou arcos, quando modeladas por triângulos lineares, requerem um grande número de elementos para a representação adequada do seu comportamento à flexão. Verificou-se também que para não-linearidades geométricas moderadas os estados de flexão não se alteram, significativamente durante a análise incremental. Logo, as malhas iniciais, geralmente compostas por um número grande de elementos, são suficientes para análise, não requerendo refinamentos adicionais.

Como trabalhos futuros, pretende-se investigar diversas possibilidades vislumbradas em [1]. Observou-se que o estimador de erro calculado sobre os lados dos elementos adequa-se naturalmente às novas estruturas de dados por lados para malhas não-estruturadas compostas por triângulos ou tetraedros. Portanto, uma continuação deste trabalho é a implementação deste estimador segundo esta nova estrutura de dados. Os desenvolvimentos em [1], apesar de tratar de algoritmos com um alto potencial de paralelização/vetorização foram implementados em linguagem C em um computador pessoal. Naturalmente, espera-se em futuro próximo implementar os procedimentos desenvolvidos em um ambiente computacional dotado da possibilidade de paralelismo e/ou vetorização. Pretende-se ainda extender à aplicação do estimador de erro baseado na média dos fluxos nos lados dos elementos à problemas de dinâmica dos fluidos computacional, onde este pode ser combinado à técnicas de refinamento e desrefinamento do tipo *h*. Em particular, a introdução de técnicas de desrefinamento se torna necessária quando se analisa problemas dependentes do tempo, seja em mecânica dos sólidos ou dos fluidos. Vale salientar que o desrefinamento não causa problemas às estratégias iterativas de solução, uma vez que este preserva as sequências regulares de malhas pré-existentes.

Mais testes são necessários para confirmar a superioridade do MGCP com precondicionador hierárquico sobre os métodos multi-malha. Particularmente não foi analisado em [1] nenhum problema envolvendo meios heterôgeneos ou com descontinuidades de material, o que pode acarretar dificuldades de convergência em ambos métodos iterativos. O desempenho do MGCP com precondicionador hierárquico em máquinas paralelas ainda está por ser investigado. Uma outra possibilidade de paralelismo pode ser a combinação do precondicionador hierárquico com técnicas de decomposição de domínios ou subestruturação.

A efetividade das técnicas propostas em [1] deverá ser verificada em problemas não-lineares dependentes da história, tais como plasticidade ou contato entre corpos deformáveis. Nestes casos, a variação da malha optima durante a análise incremental é bastante mais acentuada, o que indica um grande potencial de ganho em eficiência computacional ao se utilizar as técnicas descritas em [1].

Capítulo 2

Refinamentos de Malha Adaptativos

2.1 Introdução

Quando são empregados métodos numéricos aproximados, tais como o *método dos elementos finitos*, para obter a solução de um modelo matemático de um problema físico, três tipos de erro são gerados: os erros de *modelização* devidos as simplificações assumidas no modelo matemático, os erros de *discretização* devidos à aproximação do modelo contínuo por um modelo discreto e os erros *numéricos* devidos aos cálculos aproximados efetuados no modelo discreto.

O objetivo de uma análise adaptativa é controlar o erro de discretização procurando distribuir de forma ótima os parâmetros básicos de aproximação do método dos elementos finitos (MEF).

Uma estratégia adaptativa deve possuir duas componentes principais:

- *Estimação do erro*: isto incluí os métodos e algoritmos empregados na estimação do erro de discretização nas soluções aproximadas. Um esquema robusto de estimação do erro deveria fornecer uma boa indicação do erro em malhas grossas e uma estimação assintóticamente exata, isto é, a estimativa deveria convergir ao valor verdadeiro do erro na medida que os graus de liberdade da malha são adaptativamente incrementados.
- *Otimização da malha*: isto incluí os métodos que são empregados para predizer a distribuição ótima dos graus de liberdade da malha baseados no erro estimado, e os procedimentos utilizados para controlar as modificações da malha para obter a distribuição ótima desejada.

Dentro do MEF podem ser identificados diferentes tipos de refinamento. O mais usual é o refinamento *h* que alcança a precisão desejada diminuindo-se o tamanho dos elementos. No refinamento *p* a malha e fixada e a precisão é alcançada aumentando o grau *p* dos polinomios que definem as funções de forma. Nos refinamentos do tipo *h-p*

uma combinação apropriada dos tipos *h e p* é empregada. Existem outros tipos de refinamentos adaptativos, mas são pouco utilizados, como o método-*r* onde os nós da malha são recolocados para atender um conjunto de condições ótimas.

2.2 Erro de Discretização e Medidas do Erro

Um grande número de problemas de valores de contorno, incluindo todos os problemas de elasticidade linear, são governados pela seguinte equação diferencial linear

$$\boldsymbol{L}\,\boldsymbol{u} + \boldsymbol{p} = \boldsymbol{0} \tag{2.1}$$

no domínio Ω com condições de contorno

$$\boldsymbol{u} = \bar{\boldsymbol{u}} \quad \text{em } \Gamma_u \tag{2.2.a}$$

$$\boldsymbol{M}\,\boldsymbol{u} = \bar{\boldsymbol{t}} \quad \text{em } \Gamma_t \tag{2.2.b}$$

onde $\boldsymbol{L}$ e $\boldsymbol{M}$ são operadores diferenciais lineais, e Γ_u, Γ_t são as partições do contorno onde $\boldsymbol{u}$ ou alguma combinação de suas derivadas estão prescritas, respectivamente.

Uma aproximação por elementos finitos $\boldsymbol{u}_h$ à solução $\boldsymbol{u}$ deste problema pode ser expressa como

$$\boldsymbol{u} \approx \boldsymbol{u}_h = \boldsymbol{N}\,\hat{\boldsymbol{u}} \tag{2.3}$$

onde $\hat{\boldsymbol{u}}$ é o vetor de valores nodais e $\boldsymbol{N}$ é a matriz das funções de forma. O *erro de discretização* é definido como a diferença entre a solução exata e a solução de elementos finitos, isto é

$$\boldsymbol{e} = \boldsymbol{u} - \boldsymbol{u}_h \tag{2.4}$$

A especificação do erro local desta maneira é geralmente não conveniente para identificar a qualidade global da solução. Portanto, variadas 'normas' representando alguma quantidade integral escalar são usualmente introduzidas para medir o erro. Uma medida muito empregada é a 'norma de energia'. O erro na norma de energia é definido como

$$\|\boldsymbol{e}\| = \left(\int_\Omega \boldsymbol{e}^{\mathrm{T}}\mathbf{L}\,\boldsymbol{e}\,\mathrm{d}\Omega\right)^{1/2} \equiv \left(\int_\Omega (\boldsymbol{u}-\boldsymbol{u}_h)^{\mathrm{T}}\mathbf{L}\,(\boldsymbol{u}-\boldsymbol{u}_h)\,\mathrm{d}\Omega\right)^{1/2} \tag{2.5}$$

Quando o operador diferencial **L** é autoadjunto e a solução discreta $\boldsymbol{u}_h$ é obtida pelo método de Galerkin , tem-se

$$\|\boldsymbol{e}\|^2 = \|\boldsymbol{u}\|^2 - \|\boldsymbol{u}_h\|^2 \tag{2.6}$$

Deve-se notar que esta equação só é válida para soluções discretas $\boldsymbol{u}_h$ interpoladas por funções de forma conformes.

Para problemas de elasticidade linear plana as equações de equilíbrio e as condições de contorno podem ser escritas como

$$\boldsymbol{S}^{\mathrm{T}}\boldsymbol{\sigma} + \boldsymbol{b} = \boldsymbol{0} \tag{2.7.a}$$

$$\boldsymbol{n}^{\mathrm{T}}\boldsymbol{\sigma} = \bar{\boldsymbol{t}} \quad \text{em } \Gamma_t \tag{2.7.b}$$

$$\boldsymbol{u} = \bar{\boldsymbol{u}} \quad \text{em } \Gamma_u \tag{2.7.c}$$

onde $\boldsymbol{S}$ é o operador diferencial de primeira ordem que define as deformações $\boldsymbol{\varepsilon}$ como

$$\boldsymbol{\varepsilon} = \boldsymbol{S}\,\boldsymbol{u} \tag{2.8}$$

e $\boldsymbol{\sigma}$ representa o vetor de tensões que pode ser expresso como

$$\boldsymbol{\sigma} = \boldsymbol{D}\boldsymbol{\varepsilon} = \boldsymbol{D}\boldsymbol{S}\,\boldsymbol{u} \tag{2.9}$$

onde $\boldsymbol{D}$ é a matriz de constantes elásticas.

Para problemas de elasticidade a norma de energia dos deslocamentos $\boldsymbol{u}$ é definida pela Eq.(2.5) e fornece, após de integrar por partes

$$\|\boldsymbol{u}\| = \left(\int_{\Omega} (\mathbf{S}\boldsymbol{u})^{\mathrm{T}}\,\mathbf{D}\,(\mathbf{S}\boldsymbol{u})\;\mathrm{d}\Omega\right)^{1/2} \tag{2.10}$$

portanto o erro na norma de energia é

$$\|\boldsymbol{e}\| = \left(\int_{\Omega} (\mathbf{S}\boldsymbol{e})^{\mathrm{T}}\,\mathbf{D}\,(\mathbf{S}\boldsymbol{e})\;\mathrm{d}\Omega\right)^{1/2} \tag{2.11}$$

Os deslocamentos aproximados $\boldsymbol{u}_h$ fornecem um campo de tensões aproximado $\boldsymbol{\sigma}_h$ como

$$\boldsymbol{\sigma}_h = \boldsymbol{D}\boldsymbol{S}\,\boldsymbol{u}_h \tag{2.12}$$

logo o erro na norma de energia pose ser escrito como

$$\|\boldsymbol{e}\| = \left(\int_{\Omega} (\sigma - \sigma_h)^{\mathrm{T}}\,\mathbf{D}^{-1}\,(\sigma - \sigma_h)\;\mathrm{d}\Omega\right)^{1/2} \tag{2.13}$$

Note-se que o erro na norma de energia pode ser avaliado mediante uma somatória sobre todos os elementos como

$$\|\boldsymbol{e}\|^2 = \sum_{i=1}^{ne} (\|\boldsymbol{e}\|_i)^2 \tag{2.14}$$

onde i indica os elementos individuais Ω_i cuja união é o domínio completo Ω.

O erro relativo na norma de energia é definido como

$$\eta = \frac{\|\boldsymbol{e}\|}{\|\boldsymbol{u}\|} \tag{2.15}$$

Um valor máximo especificado $\overline{\eta}$ é usualmente empregado como critério de parada para o processo de análise adaptativa.

2.3 Estimadores de Erro para Análise Adaptativa

Como o erro exato de uma solução discreta não é conhecido por antecipado, a implementação eficiente dos procedimentos de otimização da malha conduzidos por medidas do erro necessitam de estimadores robustos para estas medidas.

As estimativas *a-posteriori* do erro estão baseadas em informação obtida durante o processo da solução. Um tipo muito eficiente de estimadores de erro estão baseados em operadores de projeção de fluxo. Neste tipo de estimadores de erro são usadas técnicas de pós-processamento que recuperam valores mais precisos das derivadas que os obtidos da solução original de elementos finitos. Para problemas de elasticidade os campos de tensões recuperadas podem ser empregados para aproximar o campo exato de tensões.

Logo, é possível substituir as tensões exatas σ pelas tensões recuperadas σ^* na Eq. (2.13) para obter um estimador de erro $\|e^*\|$ na norma de energia como

$$\|e^*\| = \left(\int_\Omega (\sigma^*-\sigma_h)\, \mathbf{D}^{-1}\, (\sigma^*-\sigma_h)\, d\Omega\right)^{1/2} \tag{2.16}$$

e assumindo que sejam empregados elementos finitos conformes, uma estimativa $\|u^*\|$ da norma de energia da solução exata pose ser obtida usando a Eq. (2.6) como

$$\|u^*\|^2 = \|u_h\|^2 + \|e^*\|^2 \tag{2.17}$$

Se fossem empregados elementos finitos não-conformes, simplesmente adota-se a estimativa $\|u^*\| = \|u_h\|$, a qual é apropriada, especialmente se o erro é pequeno.

Logo o estimador do erro relativo η^* pode ser computado como

$$\eta^* = \frac{\|e^*\|}{\|u^*\|} \tag{2.18}$$

A qualidade do estimador do erro é medida pelo *índice de efetividade* θ definido como

$$\theta = \frac{\|e^*\|}{\|e\|} \tag{2.19}$$

isto é, a razão entre as norma do erro estimado e a norma do erro exato. O estimador de erro é dito *asintóticamente exato* se θ converge à unidade com tamanhos decrescentes dos elementos da malha.

2.4 Estimador baseado nos Fluxos Médios nos Lados

Em Jouglard [1], apresenta-se um estimador de erro para triângulos lineares baseado em projeção de tensões. Tal estimador emprega coordenadas locais para

definir fluxos normais e tangenciais em cada lado dos elementos. Com estes valores calculam-se os fluxos normais e tangenciais médios em cada lado, empregando-se os valores dos

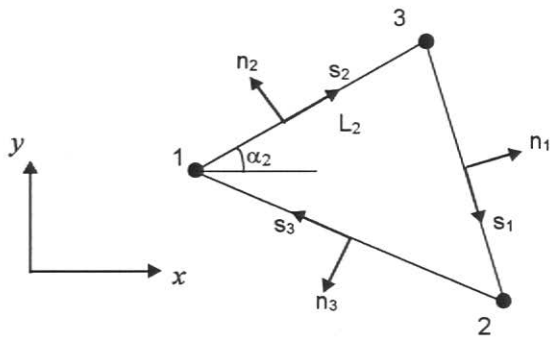


Figura 2.1. Eixos locais tangenciais e normais em cada lado.

triângulos adjacentes. Nos contornos onde existem carregamentos aplicados, o gradiente suavizado da solução pode ser forçado à adotar os valores ponderados dos fluxos prescritos. Propõem-se em [1] técnicas de ponderação dos fluxos quando há transição de lados ou cargas concentradas. Além disso, nas partes do contorno onde prescrevem-se deslocamentos é conveniente empregar-se um esquema melhorado para cálculo das tensões. O método desenvolvido em [1] consiste no cálculo de uma distribuição de tensões equivalente às reações nodais. Estas forças são obtidas por equilíbrio em todos os pontos nodais da malha, e redistribuídas usando o esquema anterior para cargas concentradas. Assim, as forças agindo em cada lado dos elementos são iguais, mesmo que o campo de tensões seja descontínuo devido à variações abruptas de material ou espessura. Deve-se notar que este tratamento para as descontinuidades pode ser útil em outras áreas, como por exemplo, escoamentos em meios porosos, onde a descrição geológica do solo, gerada tipicamente através do uso combinado de dados medidos e geoestatísticos, produz grandes variações nas propriedades físicas.

As tensões preditas pelo método anterior são contínuas somente nos pontos médios dos lados (Fig. 2.2). Esta falta de continuidade também serve como medida da qualidade da aproximação numérica, complementando as formas mais abstratas de medida, baseadas na norma de energia da solução.

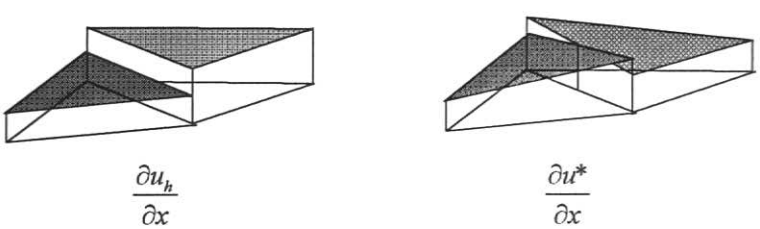


Figura 2.2. Derivadas originais e recuperadas.

Em [1] compara-se o desempenho do estimador de erro descrito acima , em problemas de elasticidade plana, com diversos outros descritos na literatura. Através da comparação de índices de efetividade e erros, conclui-se que o estimador proposto possui excelente desempenho, desde que sejam incluídos os fluxos externos, avaliados como descrito previamente.

Para problemas com grandes deslocamentos e rotações, mas pequenas deformações, usualmente conhecidos como problemas com não-linearidade geométrica, o estimador de erro descrito acima ainda pode ser empregado. Assumindo comportamento elástico, a energia de deformação da configuração deformada pode ser expressa como:

$$U = \tfrac{1}{2} \left(\int_{V_0} (\varepsilon_{NL})^T \mathbf{D} \, (\varepsilon_{NL}) \, dV_0 \right) \tag{2.20}$$

onde ε_{NL} é o tensor de deformações de Green e V_0 é o volume da configuração indeformada. Note-se que para pequenas deformações elásticas as tensões referidas à configuração indeformada podem ser aproximadas como

$$\tilde{\sigma} \approx \boldsymbol{D} \, \varepsilon_{NL} \tag{2.21}$$

Numa análise adaptativa o objetivo é obter uma descrição precisa do campo de tensões da configuração deformada. Logo para computar estimativas do erro na configuração deformada atual a seguinte norma de energia é empregada para cada elemento i :

$$\| e^* \|_i = \left(\int_{\Omega_i} (\tilde{\sigma}^* - \tilde{\sigma}_h)^T \, \mathbf{D}^{-1} \, (\tilde{\sigma}^* - \tilde{\sigma}_h) \, d\Omega_0 \right)^{1/2} \tag{2.22}$$

Assim, para o cálculo das estimativas de erro em problemas com não-linearidade geométrica são empregadas as tensões definidas pela Eq. (2.21) e a norma de energia computada como na Eq. (2.22).

Capítulo 3

Estruturas de Dados

Um dos pontos fundamentais para uma implementação adequada dos métodos adaptativos é a estrutura de dados empregada. Num refinamento do tipo *h* os elementos refinados são subdivididos regularmente.

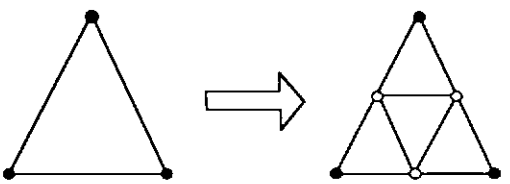


Figura 3.1. Subdivisão regular de triângulos.

Este tipo de subdivisão conduz a relações entre elementos que podem ser melhor expressadas através de estruturas de dados hierárquicas ou do tipo árvore. As malhas bidimensionais estão compostas de nós, lados e elementos. Iniciando o processo adaptativo com uma malha inicial ou de nível 0, os sucessivos nós, lados e elementos são agrupados em niveles de refinamento (Fig. 3.2).

Devido à subdivisão regular dos elementos da malha, naqueles lados que separam elementos subdivididos dos não-subdivididos podem ocorrer conexões irregulares , isto é, nós que não possuem funções de forma associadas em algum elemento adjacente. Para evitar a discontinuidade das funções de interpolação nestes lados, em [1] foi adotada uma técnica usualmente empregada com quadriláteros que consiste em forçar os valores nodais das conexões irregulares a serem coincidentes com os valores das funções de interpolação do maior dos lados adjacentes. Para obter uma transição suave em tamanho é imposta a restrição de permitir como máximo uma conexão irregular por lado (*regra 1-irregular*). Isto é equivalente a impor a condição que os niveles de dois elementos adjacentes sejam iguais ou consecutivos.

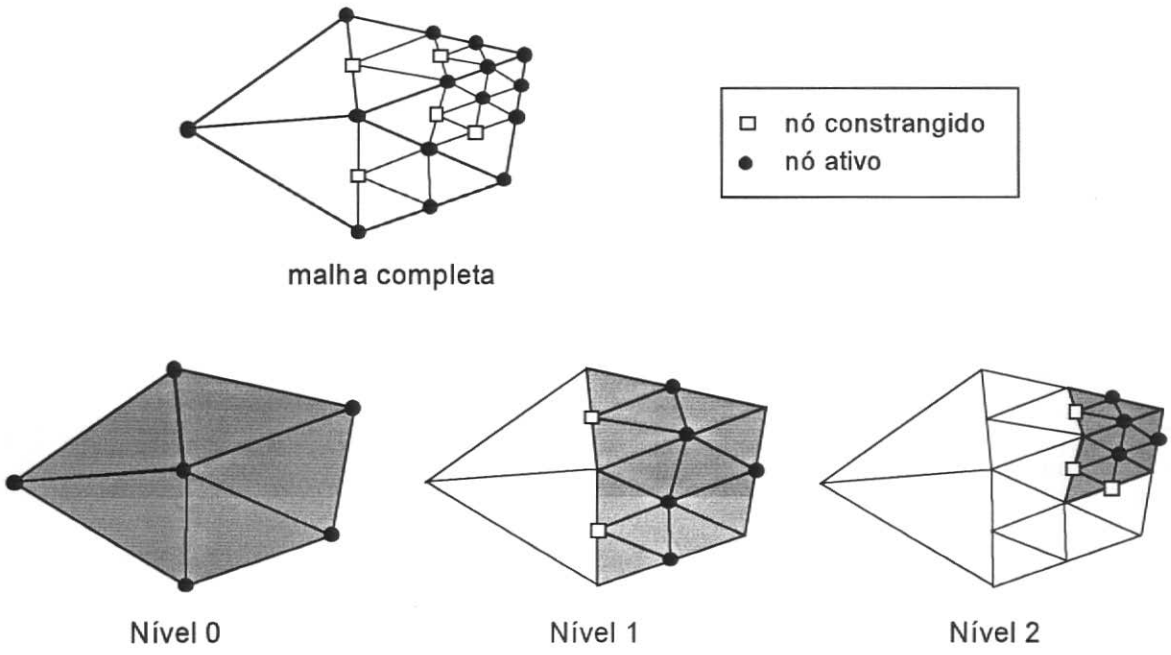


Figura 3.2. Partição de niveles da malha.

Para atender a *regra 1-irregular* em [1] é adotado um procedimento que consiste en separar os elementos de cada nível em dois tipos: *refináveis* e *não-refináveis.* Um elemento é dito *não-refinável* se ele possui algum elemento adjacente pertencente a um nivel inferior e não pode ser subdividido. Todos os elementos que não sejam *não-refináveis* são considerados *refináveis.* Para facilitar a geração dos arranjos da malha é necessario manter atualizada uma lista dos elementos *refináveis* e *não-refináveis.* O processo de refinamento consiste em fazer um lazo em ordem crescente sobre todos os niveles da malha subdividindo aqueles elementos que não satisfaçam o critério de erro. A lista de elementos *refináveis* e *não-refináveis* de cada nível é atualizada após do refinamento do nível imediato inferior. Note-se que mediante este procedimento os elementos de cada nível só podem ser subdivididos uma única vez em cada ciclo de refinamento da malha.

Os elementos de cada nível são ordenados agrupando primeiro os elementos refinados, logo os elementos refináveis não refinados e finalmente os elementos não-refináveis (Fig. 3.3).

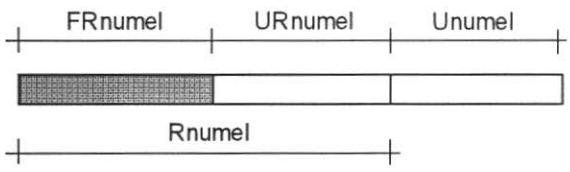


Figura 3.3. Ordenamento dos elementos de cada nível.

Onde a quantidade de elementos de cada tipo é simbolizada por:

Rnumel : *elementos refináveis de cada nível.*
Unumel : *elementos não-refináveis de cada nível.*
FRnumel : *elementos refinados de cada nível.*
URnumel : *elementos refináveis não-refinados de cada nível.*

Este ordenamento dos elementos facilita a geração dos arranjos da malha do nível seguinte após de cada refinamento.

Os lados de cada nível são ordenados em forma similar, agrupando primeiro os lados refinados, logo os elementos refináveis não refinados e finalmente os elementos não-refináveis (Fig. 3.4).

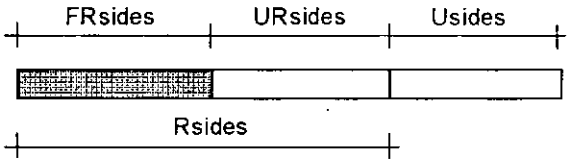


Figure 3.4. ordenamento dos lados de cada nível.

Onde a quantidade de elementos de cada tipo é simbolizada por:

Rsides : *lados refináveis de cada nível.*
Usides : *lados não-refináveis de cada nível.*
FRsides : *lados refinados de cada nível.*
URsides : *lados refináveis não-refinados de cada nível.*

Este ordenamento dos lados facilita a implementação dos solucionadores de equações multinível.

Na ref. [1] são apresentados os algoritmos empregados para efetuar as modificações das estruturas de dados para levar em conta os sucessivos refinamentos da malha.

São empregados dois arranjos básicos por nível para definir a topologia da malha. No vetor **side_nodes** são armazenados os nós extremos de cada lado, e no vetor **elm_sides** são armazenados os lados de cada elemento.

Capítulo 4

Solucionadores Multinível

4.1. Introdução

A aplicação do método dos elementos finitos implica na solução da equação matricial:

$$\boldsymbol{K}\,x = f \tag{4.1}$$

onde $\boldsymbol{K}$ é uma matriz positiva definida de dimensão $n \times n$.

Existem basicamente duas famílias de algoritmos disponíveis para a solução da equação (4.1): métodos diretos e métodos iterativos. Os solucionadores iterativos foram usados pêlos primeiros pioneiros do método dos elementos finitos no início da década de 60, mas estes solucionadores foram rapidamente descartados quando foi comprovado que para certos problemas o número de operações necessárias para atingir a convergência superava os limites teóricos. Logo os solucionadores iterativos foram abandonados e os métodos diretos baseados na fatoração triangular da matriz foram adotados devido a sua maior efetividade e robustez.

Com o incremento do poder computacional atingido com as computadoras vetoriais e paralelas o interesse nos solucionadores iterativos foi renovado devido a que estes solucionadores podem explorar o potencial desta novas arquiteturas computacionais mais eficientemente que com os métodos diretos, especialmente para problemas muito grandes onde os métodos diretos precisam de uma enorme quantidade de armazenagem o qual é freqüentemente o fator limitante para o tamanho dos problemas que podem ser analisados. Por outro lado uma das vantagens dos solucionadores iterativos empregados na solução de problemas de elementos finitos é seu escasso requerimento de armazenagem. Com estes solucionadores a matriz dos coeficientes é mantida na sua forma original e é tratada como um operador linear para computar somente produtos matriz por vetor.

No contexto de análises adaptativas, os métodos iterativos são preferíveis, devido ao fato de que se conhece uma boa aproximação da solução na malha corrente, geralmente obtida pela projeção da solução da malha anterior. Mas, a performance dos métodos iterativos é fortemente influenciada pêlos valores numéricos dos dados do problema que deve ser resolvido. Portanto um ponto importante é desenvolver métodos iterativos robustos que sejam eficientes para uma gama variada dos dados. Entre os métodos

iterativos mais eficientes encontra-se o *Método dos Gradientes Conjugados Precondicionados (MGCP)*. A eficiência do *MGCP* depende fortemente do tipo de precondicionador utilizado. Uma outra família de algoritmos iterativos muito eficientes são os *Métodos Multi-malha*, os quais requerem uma hierarquia de discretizações, isto é, empregam diversas malhas de densidade crescente na solução de um dado problema. O objetivo principal dos métodos multi-malha é se obter um método iterativo de solução onde a quantidade de trabalho computacional seja diretamente proporcional ao número de incógnitas do problema.

4.2. Precondicionadores Multinível Hierárquicos

Nos métodos iterativos são obtidas soluções aproximadas em cada passo do processo. Se substituímos estas soluções aproximadas no sistema de equações original resulta um resíduo, assim no k-ésimo passo de iteração temos:

$$\boldsymbol{r}^{(k)} = \boldsymbol{f} - \boldsymbol{K}\,\boldsymbol{x}^{(k)} \tag{4.2}$$

onde $\boldsymbol{r}^{(k)}$ é o vetor de resíduos. Uma nova aproximação $\boldsymbol{x}^{(k+1)}$ é obtida a partir da solução previa $\boldsymbol{x}^{(k)}$, o processo se repete até obter uma solução que seja precisa o suficiente.

O principal objetivo de um precondicionador é melhorar a taxa de convergência de um algoritmo de relaxação. Ao invés de solucionar o sistema original (4.1) solucionamos

$$\boldsymbol{K}\,\boldsymbol{B}^{-1}\,\boldsymbol{y} = \boldsymbol{f} \tag{4.3}$$

onde

$$\boldsymbol{B}^{-1}\,\boldsymbol{y} = \boldsymbol{x} \tag{4.4}$$

A matriz $\boldsymbol{B}$ é chamada *matriz de precondicionamento*, e deve ser escolhida de maneira de solucionar rápidamente o sistema de equações (4.4) e deve ser uma boa aproximação à matriz $\boldsymbol{K}$. Um dos precondicionadores mais simples é o precondicionador *diagonal*, neste caso a matriz $\boldsymbol{B}$ é uma matriz diagonal cujos coeficientes diagonais são aqueles da matriz $\boldsymbol{K}$.

Um dos precondicionadores mais eficientes são os *precondicionadores multinível*, os quais requerem uma seqüência de refinamentos (uniformes ou adaptativos) de uma malha inicial grossa.

Se a base nodal padrão do método dos elementos finitos é substituída por bases hierárquicas quando uma malha de um dado nível é refinada, a discretização resultante preserva todas as informações dos níveis precedentes. Se consideramos uma sucesão de malhas geradas por refinamento, para cada nível de refinamento o sistema de equações deve ser inteiramente reavaliado, isto é:

nível 0
$$\boldsymbol{K}_{00}^{(0)}\,\boldsymbol{x}_0 = \boldsymbol{f}_0^{(0)} \tag{4.5.a}$$

nível 1
$$\begin{bmatrix} \boldsymbol{K}_{00}^{(1)} & \boldsymbol{K}_{01}^{(1)} \\ \boldsymbol{K}_{10}^{(1)} & \boldsymbol{K}_{11}^{(1)} \end{bmatrix} \begin{Bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_1 \end{Bmatrix} = \begin{Bmatrix} \boldsymbol{f}_0^{(1)} \\ \boldsymbol{f}_1^{(1)} \end{Bmatrix} \tag{4.5.b}$$

nível 2

$$\begin{bmatrix} K_{00}^{(2)} & K_{01}^{(2)} & K_{02}^{(2)} \\ K_{10}^{(2)} & K_{11}^{(2)} & K_{12}^{(2)} \\ K_{20}^{(2)} & K_{21}^{(2)} & K_{22}^{(2)} \end{bmatrix} \begin{Bmatrix} x_0 \\ x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} f_0^{(2)} \\ f_1^{(2)} \\ f_2^{(2)} \end{Bmatrix} \tag{4.5.c}$$

onde o número entre parenteses indica o nível onde cada matriz ou vetor foi formada.

Mas, é possivél usar funções de forma *"hierárquicas"* cujas amplitudes não estão diretamente associadas com valores nodais. Uma característica importante das bases hierárquicas é que quando um nível é refinado a nova discretização preserva a informação das discretizações previas. O uso das bases hierárquicas resulta na seguinte seqüência de aproximações:

nível 0

$$\hat{K}_{00}^{(0)} \hat{x}_0 = \hat{f}_0^{(0)} \tag{4.6.a}$$

nível 1

$$\left[\begin{array}{c|c} \hat{K}_{00}^{(0)} & \hat{K}_{01}^{(1)} \\ \hline \hat{K}_{10}^{(1)} & \hat{K}_{11}^{(1)} \end{array}\right] \begin{Bmatrix} \hat{x}_0 \\ \hat{x}_1 \end{Bmatrix} = \begin{Bmatrix} \hat{f}_0^{(0)} \\ \hat{f}_1^{(1)} \end{Bmatrix} \tag{4.6.b}$$

nível 2

$$\left[\begin{array}{cc|c} \hat{K}_{00}^{(0)} & \hat{K}_{01}^{(1)} & \hat{K}_{02}^{(2)} \\ \hat{K}_{10}^{(1)} & \hat{K}_{11}^{(1)} & \hat{K}_{12}^{(2)} \\ \hline \hat{K}_{20}^{(2)} & \hat{K}_{21}^{(2)} & \hat{K}_{22}^{(2)} \end{array}\right] \begin{Bmatrix} \hat{x}_0 \\ \hat{x}_1 \\ \hat{x}_2 \end{Bmatrix} = \begin{Bmatrix} \hat{f}_0^{(0)} \\ \hat{f}_1^{(1)} \\ \hat{f}_2^{(2)} \end{Bmatrix} \tag{4.6.c}$$

Para cada nível mostrado acima pode-se observar que as matrizes e vetores gerados em niveles prévios não se modificam e portanto não precissam ser calculadas novamente. Note-se que devido a que as variaveis do nível inicial são iguais para ambas as bases, as matrizes do nível inicial são iguais:

$$\hat{K}_{00}^{(0)} = K_{00}^{(0)} \tag{4.7}$$

Logo é possível definir uma matriz de *precondicionamento multinível hierárquico* $\hat{B}$ como

$$\hat{B} = \begin{bmatrix} (LDL^{\mathrm{T}})_{00}^{(0)} & 0 & 0 \\ 0 & \hat{D}_{11}^{(1)} & 0 \\ 0 & 0 & \hat{D}_{22}^{(2)} \end{bmatrix} \tag{4.8}$$

onde $(LDL^{\mathrm{T}})_{00}^{(0)}$ representa a fatorização de Crout da matriz $K_{00}^{(0)}$, e $\hat{D}_{11}^{(1)}$, $\hat{D}_{22}^{(2)}$ são os elemntos diagonais das matrizes $\hat{K}_{11}^{(1)}$, $\hat{K}_{22}^{(2)}$, respectivamente.

4.3. Métodos Multi-malha Adaptativos

Os métodos multi-malha são uma família de rápidos algoritmos iterativos cuja idea básica consiste em efetuar relaxações sobre uma malha fina combinadas com a solução sobre uma malha grossa para reduzir as freqüências altas e bajas do erro associadas com a solução aproximada de um problema discreto. Estes métodos requerem uma hierarquia de discretizações, isto é, são empregadas malhas de densidade crescente as quais, no método multi-malha convencional, são geradas mediante sucesivos refinamentos uniformes de uma malha grossa inicial.

Nos métodos multi-malha adaptativos combinam-se as ideas básicas do método multi-malha convencional com um esquema de refinamento adaptativo para gerar a hierarquía requerida das malhas.

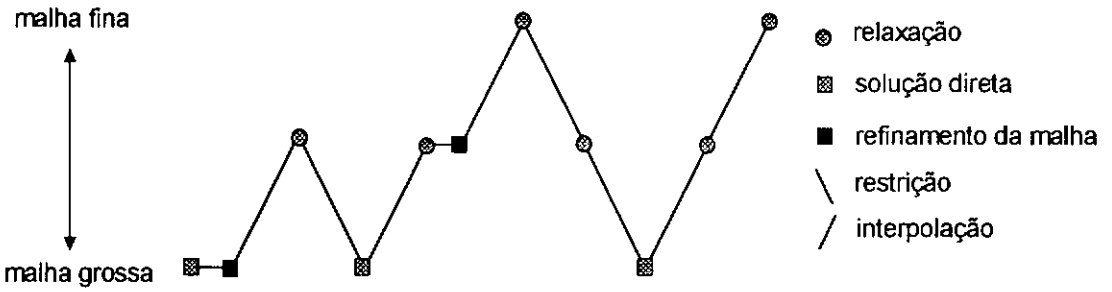


Figura 4.1. Representação gráfica do Método Multi-malha Adaptativo.

Num algoritmo multi-malha adaptativo, um refinamento de malha é requerido depois de obter a solução da eq. (4.1) para produzir uma nova malha fina. Os refinamentos da malha acabam quando o estimador de erro escolhido indica que a solução foi obtida dentro dos limites especificados pelo usuário para o erro de discretização.

4.4. Critério de Convergência

Um procedimento muito usado para acabar um método iterativo de solução da eq. (4.1), é calcular o resíduo $\boldsymbol{r}^{(k)}$ associado com a solução aproximada $\boldsymbol{x}^{(k)}$ depois de k ciclos e parar de iterar quando

$$\|\boldsymbol{r}^{(k)}\| / \|\boldsymbol{f}\| < \varepsilon \tag{4.9}$$

onde $\| \cdot \|$ é a norma Euclideana e ε é uma tolerância fornecida pelo usuário. Usando este critério pode-se usualmente obter uma solução dentro dos límites de precisão do computador.

Aos fines de engenharia o objetivo de uma análise numérica é obter uma solução discreta para um sistema de equações diferenciais parciais com um erro de discretização aceitável. Usualmente o erro de discretização é medido na norma de energia $\| \cdot \|_K$ definida para um vetor arbitrário de erro $\boldsymbol{e}$ como

$$\| \boldsymbol{e} \|_K = (\boldsymbol{e}^t \boldsymbol{K} \boldsymbol{e})^{1/2} \tag{4.10}$$

Note-se que esta definição é equivalente à definição dada pela eq. (2.5) para a norma de energia do erro.

O vetor erro $\boldsymbol{e}$ representa a diferencia entre a solução discreta obtida do problema discretizado e a solução exata do problema contínuo. Levando em conta que a solução exata do problema contínuo não está geralmente disponível, várias estimativas são empregadas na prática fornecendo um erro estimado $\boldsymbol{e}^*$. Estas estimativas estão baseadas na solução do sistema de equações (4.1), e geralmente uma solução aproximada, ainda que precisa, é suficiente para fornecer uma adequada estimativa do erro de discretização. Logo, uma solução aproximada $\boldsymbol{x}^{(k+1)}$ pode ser considerada aceitavél se

$$\| \boldsymbol{e}^* \|_K < \eta \| \boldsymbol{x} \|_K \approx \eta \| \boldsymbol{x}^{(k+1)} \|_K \tag{4.11}$$

onde η é a tolerância do erro de discretização na norma de energia.

A verificação do critério anterior é custosa. Uma outra alternativa é verificar a diferencia entre duas soluções sucessivas. Isto é, se $\boldsymbol{x}^{(k)}$ and $\boldsymbol{x}^{(k+1)}$ são duas soluções sucessivas do processo iterativo verificamos se a seguinte desigualdade é atendida

$$\| \boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)} \|_K < \varepsilon_U \| \boldsymbol{x}^{(k+1)} \|_K \tag{4.12}$$

onde ε_U é a tolerância numérica na norma de energia.

Deve-se notar que a variação do lado esquerdo da desigualdade pode ser considerada monotónica com as iterações e existe evidencia numérica que a norma de energia relativa do incremento pode ser consideada aproximadamente igual ao erro relativo na norma de energia da solução numérica, isto é

$$\| \boldsymbol{x}^{(k+1)} - \boldsymbol{x}_h \|_K / \| \boldsymbol{x}_h \|_K \approx \| \boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)} \|_K / \| \boldsymbol{x}^{(k+1)} \|_K < \varepsilon_U \tag{4.13}$$

onde $\boldsymbol{x}_h$ é a solução numérica exata para esta malha.

Logo, um valor tentativo para a tolerância numérica ε_U poderia ser $\varepsilon_U = \eta$, mas como a precisão da solução aproximada $\boldsymbol{x}^{(k+1)}$ afeta a precisão do estimador do erro, em geral um valor muito mas pequeno deve ser adotado como fica demonstrado em [1].

4.5. Método de Newton-Raphson Multinível para Problemas Não-lineares.

Neste método um solucionador multinível 0 é usado dentro do ciclo de iterações de Newton. Considerando carregamento proporcional controlado por um parâmetro de cargas λ, para cada incremento k do carregamento a equação linear

$$\boldsymbol{K}_T^{(k)} \, \Delta \boldsymbol{x}_k^{(m+1)} = \lambda_k \boldsymbol{f}_0 - \boldsymbol{f}_i^{(m)} \tag{4.14}$$

deve ser solucionada para $\Delta \boldsymbol{x}_k^{(m+1)}$ usando o solucionador multinível, e um novo conjunto de deslocamentos

$$\boldsymbol{x}_k^{(m+1)} = \boldsymbol{x}_k^{(m)} + \Delta \boldsymbol{x}_k^{(m+1)} \tag{4.15}$$

são calculados. Este procedimento é repetido até que as forças internas estejam em equilíbrio com as forças externas dentro de alguma tolerância especificada. O subíndice k indica o passo de carga, entanto que o supraíndice m indica o numero de iteração de Newton para essa carga. O parâmetro λ_k representa o nível de carga aplicada e f_0 é o carregamento de referencia. A matriz $\boldsymbol{K}_T^{(k)}$ é a matriz de rigidez tangente computada no inicio do passo para o método *modificado* de Newton-Raphson, ou é atualizada em cada iteração de Newton para o método *completo* de Newton-Raphson. O vetor $\Delta \boldsymbol{x}_k^{(m+1)}$ é o incremento de deslocamentos, $\boldsymbol{x}_k^{(m)}$ e $\boldsymbol{x}_k^{(m+1)}$ são os deslocamentos antes e depois da iteração de Newton atual, e $\boldsymbol{f}_i^{(m)}$ são as forças internas associadas com os deslocamentos $\Delta \boldsymbol{x}_k^{(m)}$.

A precisão do método multinível de Newton-Raphson depende de vários critérios de convergência. Dois laços iterativos podem ser identificados: um laço externo de iterações de Newton-Raphson e um laço interno de iterações no solucionador multinível. O teste de convergencia aplicado ao laço externo determina a precisão da solução do problema não-linear, e examina a razão da norma Euclideana dos deslocamentos incrementais em relação com os deslocamentos acumulados

$$\Delta \boldsymbol{x}_k^{(m)} < \varepsilon_0 \sum_{i=1}^{m} \Delta \boldsymbol{x}_k^{(i)} \tag{4.16}$$

onde ε_0 é a tolerância do laço externo especificada pelo usuário.

Deve-se notar que quando combinado com adaptatividade devemos verificar se a malha no inicio do incremento de carga atende o erro de discretização ao final das iterações de Newton. Se o erro de discretização não é satisfeito refinamos a malha e as iterações de Newton-Raphson são repetidas desde o inicio do passo com esta nova malha.

Capítulo 5

Resultados Numéricos

5.1. Comparações de Solucionadores Multinível

Usamos a seguinte convenção para os diferentes solucionadores empregados nas comparações:

PCG-H : gradientes conjugados com precondicionador hierárquico.
PCG-D : gradientes conjugados com precondicionador diagonal.
MG-H : multi-malha com PCG hierárquico para as relaxações.
MG-D : multi-malha com PCG diagonal para as relaxações.

Em todos os solucionadores é usado o critério de energia para convergência com uma tolerância $\varepsilon_U = 0.001$ (0.1% na norma de energia), e o teste de convergência é feito em cada iteração sobre a malha fina.

Todas as comparações foram efetuadas numa PC 486 (100 mhz, 8MB Ram).

5.2. Região em Forma de L em Tensão Plana

Foi analisado o problema standard do dominio plano em forma de L cuja geometria, cargas, condições de contorno e malha inicial são mostradas na Fig. 5.1.

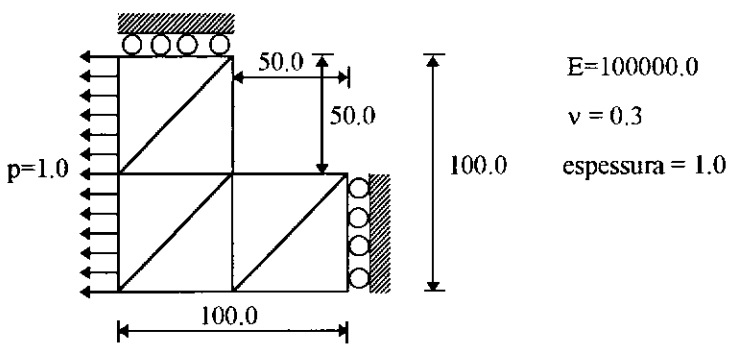
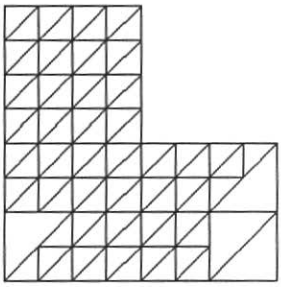
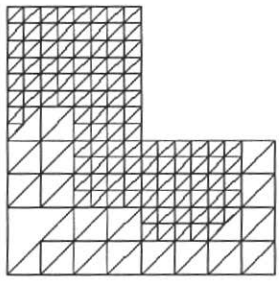


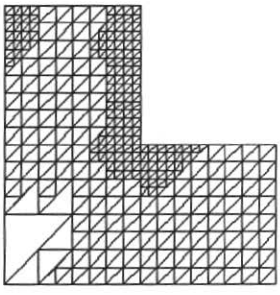
Figura 5.1 - Problema do dominio em forma de L.



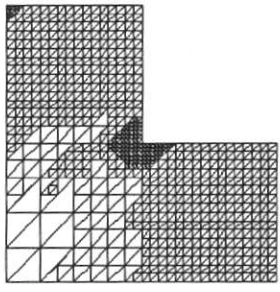
MALHA 2 η = 34.46 %
GDL = 118



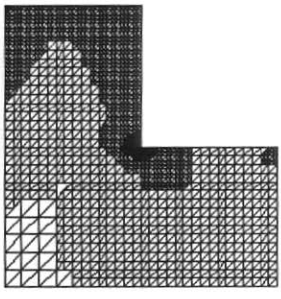
MALHA 3 η = 21.99 %
GDL = 318



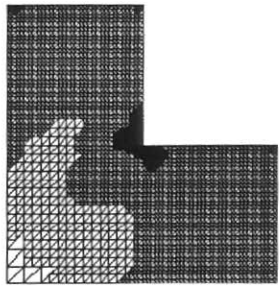
MALHA 4 η = 15.53 %
GDL = 686



MALHA 5 η = 10.46 %
GDL = 1502



MALHA 6 η = 7.18 %
GDL = 3086



MALHA 7 η = 4.81 %
GDL = 6222

Figura 5.2. Estimador do erro baseado nos fluxos médios dos lados.

Neste exemplo a malha inicial foi adaptativamente refinada usando uma tolerância para o erro de discretização $\eta = 5\%$, e foi utilizado o estimador de erro baseado nos fluxos médios nos lados, resultando sete niveles de refinamento que são mostrados na Fig. 5.2.

Na Figura 5.3 é mostrado o tempo total de CPU empregado por cado solucionador. Pode-se observar que solucionador PCG diagonal foi o mais lento, e o solucionador PCG hierárquico foi o mais rápido. Ambos os algoritmos multi-malha foram mais lentos que o solucionador PCG hierárquico, e em particular para este exemplo o solucionador MG-D levó 34% mais tempo que o solucionador PCG-H e o solucionador MG-H levó 23% mais tempo que o solucionador PCG-H. Na Tabela 5.1 são mostrados os números totais de iterações empregados por cada solucionador na malha fina de cada nivel, onde os números em parénteses nas colunas dos solucionadores multi-malha são os números de ciclos multi-malha empregados.

Pode-se observar que o número de iterações em cada malha permanece aproximadamente constante para o solucionador PCG-H, e o mesmo pode-se dizer do número de ciclos multi-malha para o solucionador MG-H. Isto pode ser atribuído à combinação apropriada de precondicionadores hierárquicos, critérios de refinamento adaptativo e a tolerância numérica escolhida para as iterações. O incremento de tempo para o solucionador MG-H sobre o solucionador PCG-H pode ser atribuído ao fato que os precondicionadores são mais efetivos para os solucionadores PCG que para os solucionadores multi-malha, pois as propriedades de suavização das relaxações multi-malha intermediárias reduzen a efetividade dos precondicionadores mais elaborados. Isto pode ser observado na comparação dos tempos dos solucionadores MG-H e MG-D, onde o ganho em efetividade do precondicionador hierárquico não é tão pronunciado como no caso dos solucionadores PCG.

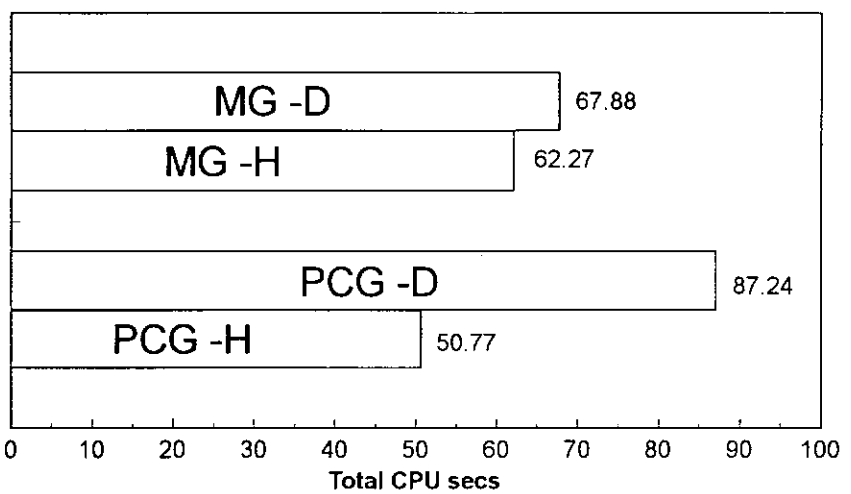


Figura 5.3 - Tempo total de CPU para análise linear adaptativa.

Tabela 5.1 - Número de iterações na malha fina de cada nível.

	ITERAÇÕES NA MALHA FINA DE CADA NIVEL			
MALHA (GDL)	PCG - H	PCG - D	MG - H	MG - D
1 (38)	12	22	15 (2)	31 (4)
2 (118)	19	44	20 (2)	39 (4)
3 (318)	21	90	18 (2)	31 (4)
4 (618)	21	119	17 (2)	18 (2)
5 (1502)	20	129	11 (2)	11 (2)
6 (3086)	19	35	11 (2)	9 (1)
7 (6222)	16	29	8 (1)	8 (1)
TEMPO TOTAL	50.77 secs.	87.24 secs.	62.27 secs.	67.88 secs.

Na tabela 5.2 os requerimentos de memoria de cada solucionador são mostrados. Deve-se notar que ambos os solucionadores PCG requerem a mesma quantidade de armazenamento devido a que um solucionador direto é usado na a malha grossas inicial para iniciar o processo adaptativo. Os números em parenteses nas colunas dos solucionadores multi-malha indicam o incremento proporcional de memoria com relação ao solucionador PCG para o mesmo nivel de refinamento.

Tabela 5.2 - Memoria necessaria para cada nível de refinamento.

	MEMORIA (Bytes)		
MALHA (GDL)	PCG	MG - H	MG - D
1 (38)	7252 (1.00)	7812 (1.08)	8244 (1.14)
2 (118)	22458 (1.00)	24330 (1.08)	25738 (1.15)
3 (318)	59400 (1.00)	64920 (1.09)	68952 (1.16)
4 (618)	127648 (1.00)	142896 (1.12)	153264 (1.20)
5 (1502)	278026 (1.00)	320954 (1.15)	348426 (1.25)
6 (3086)	569448 (1.00)	674872 (1.19)	739928 (1.30)
7 (6222)	1145818 (1.00)	1409706 (1.23)	1566538 (1.37)

5.3. Chapa sob Tração Simples com uma Fratura

Neste exemplo analizamos uma chapa sob tração simples com uma fratura no centro. Devido à simetría somente um quarto do plano é considerado. A geometría, cargas, condições de contorno e a malha inicial são mostrados na Fig. 5.4.

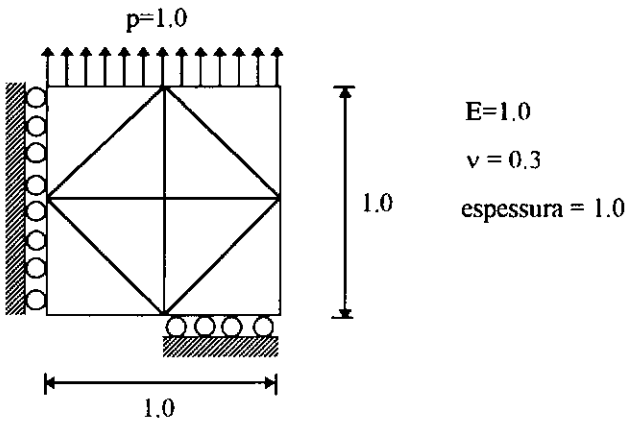


Figura 5.4. Chapa sob tração simples com uma fratura no centro.

São considerados grandes deslocamentos mas pequenas deformações. Foi feita uma analise incremental adaptativa não-linear com dez incrementos de carga ($\Delta p=0.01$) e para um tolerância no erro de discretização de 17% , iniciando o processo adaptativo com a malha grossa da Fig. 5.4. Na Fig. 5.5 pode ser observada a evolução da abertura da fratura. Um número promedio de cinco iterações de Newton-Raphson foram necessárias por paso, e dois adaptações de malha foram feitas no primeiro e segundo paso, cujas respectivas malhas são mostradas nas Figs. 5.6 e 5.7. Na Fig. 5.8 pode-se observar a configuração final deformada.

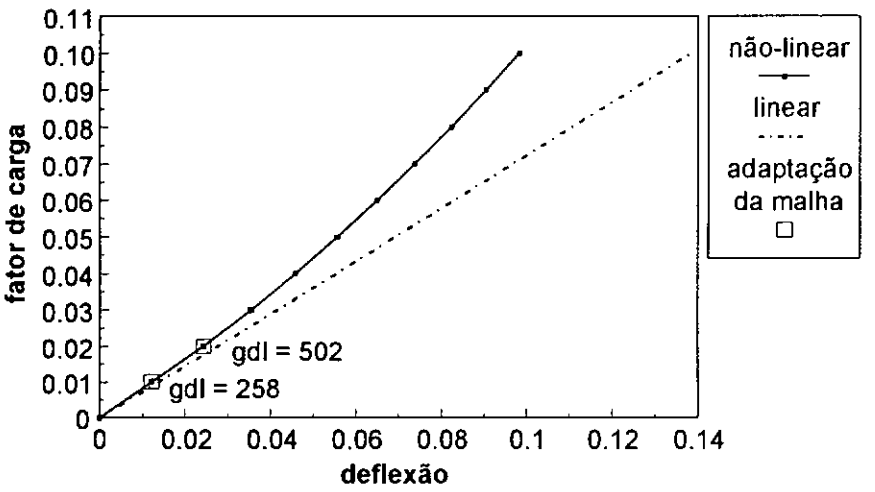


Figura 5.5. Evolução da abertura da fratura.

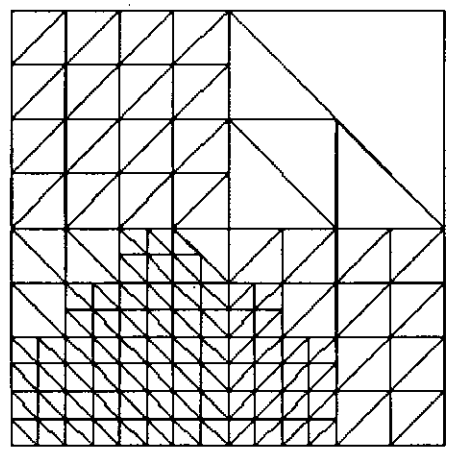


Figura 5.6. Malha com 258 graus de liberdade (gdl).

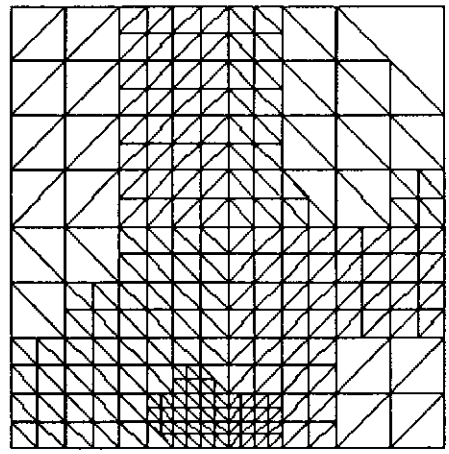


Figura 5.7. Malha com 502 graus de liberdade (gdl).

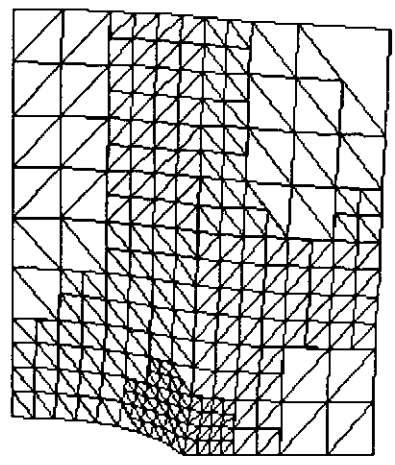


Figura 5.8. Configuração final deformada.

Capítulo 6

Conclusões

No capítulo 2 foi apresentado um estimador de erro para triângulos baseado no fluxo médio nos lados. As predições de tensões são somente contínuas nos pontos médios dos lados. Esta falta de continuidade também serve como medida da qualidade da aproximação numérica, complementando as formas mais abstratas de medida, baseadas na norma de energia da solução.

Como trabalhos futuros, pretende-se investigar diversas possibilidades vislumbradas em [1]. Observou-se que o estimador de erro calculado sobre os lados dos elementos adequa-se naturalmente às novas estruturas de dados por lados para malhas não-estruturadas compostas por triângulos ou tetraedros. Portanto, uma continuação deste trabalho é a implementação deste estimador segundo esta nova estrutura de dados. Os desenvolvimentos em [1], apesar de tratar de algoritmos com um alto potencial de paralelização/vetorização foram implementados em linguagem C em um computador pessoal. Naturalmente, espera-se em futuro próximo implementar os procedimentos desenvolvidos em um ambiente computacional dotado da possibilidade de paralelismo e/ou vetorização. Pretende-se ainda extender à aplicação do estimador de erro baseado na média dos fluxos nos lados dos elementos à problemas de dinâmica dos fluidos computacional, onde este pode ser combinado à técnicas de refinamento e desrefinamento do tipo *h*. Em particular, a introdução de técnicas de desrefinamento se torna necessária quando se analisa problemas dependentes do tempo, seja em mecânica dos sólidos ou dos fluidos. Vale salientar que o desrefinamento não causa problemas às estratégias iterativas de solução, uma vez que este preserva as sequências regulares de malhas pré-existentes.

Mais testes são necessários para confirmar a superioridade do MGCP com precondicionador hierárquico sobre os métodos multi-malha. Particularmente não foi analisado em [1] nenhum problema envolvendo meios heterôgeneos ou com descontinuidades de material, o que pode acarretar dificuldades de convergência em ambos métodos iterativos. O desempenho do MGCP com precondicionador hierárquico em máquinas paralelas ainda está por ser investigado. Uma outra possibilidade de paralelismo pode ser a combinação do precondicionador hierárquico com técnicas de decomposição de domínios ou subestruturação.

A efetividade das técnicas propostas em [1] deverá ser verificada em problemas não-lineares dependentes da história, tais como plasticidade ou contato entre corpos deformáveis. Nestes casos, a variação da malha optima durante a análise incremental é bastante mais acentuada, o que indica um grande potencial de ganho em eficiência computacional ao se utilizar as técnicas descritas em [1].

Referências

[1] C.E. Jouglard, "A Multilevel Adaptive Approach for the Solution of Non-Linear Finite Element Problems", Research Report EC-ES001/96, Department of Civil Engineering, COPPE/UFRJ, 1996.

[2] Papadrakakis, M., Solving large-scale linear problems in solid and structural mechanics, in: M. Papadrakakis, ed., *Solving Large-scale Problems in Mechanics*, John Wiley & Sons Ltd, Chichester, England (1993).

[3] Briggs, W.L., *A Multigrid Tutorial*. SIAM, Philadelphia (1987).

[4] Bank, R.E. and Sherman, A.H., An adaptive, multilevel method for elliptic boundary value problems, *Computing*, **26**, 91-105 (1981).

[5] Rivara, M.C., Algorithms for refining triangular grids suitable for adaptive and multigrid techniques, *Int. J. Numer. Methods Eng.*, **20**, 745-756 (1984).

[6] Rivara, M.C., Design and data structure of fully adaptive, multigrid, finite-element software, *ACM Trans. Math. Software*, **10**, 242-264 (1984).

[7] Rheinboldt, W.C. and Mesztenyi, Ch.K., On a data structure for adaptive finite element mesh refinements, *ACM Trans. Math. Software*, **6**, 166-187 (1980).

[8] Devloo, Ph., Oden, J.T. and Stroboulis, T., Implementation of an adaptive refinement technique for the SUPG algorithm, *Comput. Meths. Appl. Mech. Eng.*, **61**, 339-358 (1987).

[9] Löhner, R., An adaptive finite element scheme for transient problems in CFD, *Comput. Meths. Appl. Mech. Eng.*, **61**, 323-338 (1987).

[10] Babuska, I., The problem of modeling the elastomechanics in engineering, *Comput. Meths. Appl. Mech. Eng.*, **82**, 155-182 (1990).

[11] Bathe, K.J., Lee, N.S. and Bucalem, M., On the use of hierarchical models in engineering analisys, *Comput. Meths. Appl. Mech. Eng.*, **82**, 5-26 (1990).

[12] Bathe, K.J., *Finite Elements Procedures in Engineering Analysis*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey (1982).

[13] Zienkiewicz O.C., and Taylor R.L., *The Finite Element Method*, 4th edn., Vol.1, McGraw-Hill, New York, (1989).

[14] Hughes, T.J.R., *The Finite Element Method,* Prentice-Hall, Inc., Englewood Cliffs, New Jersey (1987).

[15] Babuska, I., and Rheinboldt, W.C., Error estimates for adaptive finite element computations, *SIAM J. Numer. Anal.*, **15 (4)**, 736-754 (1978).

[16] Babuska, I., and Rheinboldt, W.C., A-posteriori error estimates for the finite element method, *Int. J. Numer. Methods Eng.*, **12**, 1597-1615 (1978).

[17] Ainsworth, M. and Craig, A., A-posteriori error estimators in the finite element method, *Numer. Math.*, **60**, 429-463 (1992).

[18] Ainsworth, M. and Oden , J.T., A unified approach to a posteriori error estimation using element residual methods, *Numer. Math.*, **65**, 23-50 (1993).

[19] Babuska, I., Duran, R. and Rodriguez, R., Analysis of the efficiency of an a-posteriori error estimator for linear triangular finite elements, *SIAM J. Numer. Anal.*, **29 (4)**, 947-964 (1992).

[20] Babuska, I., and Yu, D., Asymptotically exact a-posteriori error estimator for biquadratic elements, *Finite Elements Anal. Design*, **3**, 341-354 (1987).

[21] Bank, R.E. and Weiser, A., Some a-posteriori error estimates for elliptic partial differential equations, *Math. Comp.*, **44**, 283-301 (1985).

[22] Durán, R., Muschietti, M.A. and Rodriguez R., On the asymptotic exactness of the error estimator for linear triangular elements, *Numer. Math.*, **59**, 78-88 (1991).

[23] Kelly, D.W., The self-equilibration of residuals and complementary a-posteriori error estimates in the finite element method, *Int. J. Numer. Methods Eng.*, **20**, 1491-1506 (1984).

[24] Ladeveze, P. and Leguillon, Error estimation procedure in the finite element method and applications, *SIAM J. Numer. Anal.*, **20**, 485-509 (1983).

[25] Oden, J.T., Demkowicz, L., Rachowicz, W. and Westermann, T.A., Toward a universal h-p adaptive finite element strategy, Part 2: a-posteriori error estimates, *Comput. Meths. Appl. Mech. Eng.*, 77, 113-180 (1989).

[26] Verfürth, R., A-posteriori error estimators for the Stokes equation, *Numer. Math.*, **55**, 309-325 (1989).

[27] Szabo, B.A., Mesh design for the p-version of the finite element method, *Comput. Meths. Appl. Mech. Eng.*, **55**, 181-197 (1986).

[28] Demkowicz, L., Devloo, P. and Oden, J.T., On h-type mesh-refinement strategy based on minimization of interpolation error, *Comput. Meths. Appl. Mech. Eng.*, **53**, 67-89 (1985).

[29] Eriksson, K. and Johnson C., An adaptive finite element method for linear elliptic problems, *Math. Comp.*, **50**, 361-383 (1988).

[30] Zienkiewicz, O.C., and Zhu, J.Z., A simple error estimator and adaptive procedure for practical engineering analysis, *Int. J. Numer. Methods Eng.*, **24**, 337-357 (1987).

[31] Zienkiewicz, O.C., and Zhu, J.Z., The superconvergent patch recovery and a posteriori error estimates. Part 1: The recovery technique, *Int. J. Numer. Methods Eng.*, **33**, 1331-1364 (1992).

[32] Zienkiewicz, O.C., and Zhu, J.Z., The superconvergent patch recovery and a posteriori error estimates. Part 2: Error estimates and adaptivity, *Int. J. Numer. Methods Eng.*, **33**, 1365-1382 (1992).

[33] Wiberg, N.E., Abdulwahab, F. and Ziukas, S., Enhanced superconvergent patch recovery incorporating equilibrium and boundary conditions, *Int. J. Numer. Methods Eng.*, **37**, 3417-3440 (1994).

[34] Blacker, T. and Belytschko, T., Superconvergent patch recovery with equilibrium and conjoint interpolant enhancements, *Int. J. Numer. Methods Eng.*, **37**, 517-546 (1994).

[35] Babuska, I., Strouboulis, T., Upadhyay, C.S., Gangaraj, S.K. and Copps, K., Validation of a posteriori error estimators by numerical approach, *Int. J. Numer. Methods Eng.*, **37**, 1073-1123 (1994).

[36] Strouboulis, T. and Haque, K.A., Recent experiences with error estimation and adaptivity, Part I: review of error estimators for scalar elliptic problems, *Comput. Meths. Appl. Mech. Eng.*, **97**, 399-436 (1992).

[37] Strouboulis, T. and Haque, K.A., Recent experiences with error estimation and adaptivity, Part II: error estimation for h-adaptive approximations on grids of triangles and quadrilaterals, *Comput. Meths. Appl. Mech. Eng.*, **100**, 359-430 (1992).

[38] Stein, E., and Ahmad, R., On stress computation in finite elements models based upon displacement approximations, *Comput. Meths. Appl. Mech. Eng.*, **4**, 81-96 (1974).

[39] Oden, J.T., and Brauchli, H.T., On the calculation of consistent stress distribution in finite elements calculations, *Int. J. Numer. Methods Eng.*, **3**, 317-325 (1971).

[40] Oden, J.T., and Reddy, J.N., Note on an approximate method for computing consistent conjugate stresses in elastic finite elements, *Int. J. Numer. Methods Eng.*, **6**, 55-61 (1973).

[41] Hinton, E. and Campbell, J.S., Local and global smoothing of discontinuos finite element functions using a least squares method, *Int. J. Numer. Methods Eng.*, **8**, 461-480 (1974).

[42] Barlow, J., Optimal stress locations in finite element models, *Int. J. Numer. Methods Eng.*, **10**, 243-251 (1976).

[43] Mackinnon, R.J. and Carey, G.F., Superconvergent derivatives: ATaylor series analysis, *Int. J. Numer. Methods Eng.*, **28**, 489-509 (1989).

[44] Zienkiewicz, O.C., and Zhu, J.Z., The SPR recovery and boundaries, *Int. J. Numer. Methods Eng.*, **37**, 3195-3201 (1994).

[45] Babuska, I., and Rheinboldt, W.C., Analysis of optimal finite-element meshes in R^1, *Math. Comp.*, **33**, 435-463 (1979).

[46] Diaz, A.R., Kikuchi, N. and Taylor, J.E., A method of grid optimization for finite element methods, *Comput. Meths. Appl. Mech. Eng.*, **41**, 29-45 (1983).

[47] Zhu, J.Z. and Zienkiewicz, O.C., Adaptive techniques in the finite element method, *Commun. Appl. Numer. Methods*, **4**, 197-204 (1988).

[48] Lo, S.H., A new mesh regeneration scheme for arbitrary planar domains, *Int. J. Numer. Methods Eng.*, **21**, 1403-1426 (1985).

[49] Peraire, J., Vahdati, M., Morgan, K. and Zienkiewicz, O.C., Adaptive remeshing for compressible flow computations, *J. Comp. Phys.*, **72**, 449-466 (1987).

[50] Jin, H. and Wiberg, N.E., Two dimensional mesh generation, adaptive remeshing and refinement, *Int. J. Numer. Methods Eng.*, **29**, 1501-1526 (1990).

[51] Lo, S.H., Automatic mesh generation and adaptation using contours, *Int. J. Numer. Methods Eng.*, **31**, 689-707 (1991).

[52] Oden, J.T., Demkowicz, L., Rachowicz, W. and Hardy, O., Toward a universal h-p adaptive finite element strategy, Part 1: Constrained approximation and data structure, *Comput. Meths. Appl. Mech. Eng.*, 77, 79-112 (1989).

[53] Fung, Y. C., *Foundations of Solid Mechanics*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey (1965).

[54] Crisfield, M.A., *Non-Linear Finite Element Analysis of Solids and Structures*, John Wiley and Sons Ltd, (1991)

[55] Zienkiewicz, O.C., and Zhu, J.Z., Error estimates and adaptive refinements for plate bending problems, *Int. J. Numer. Methods Eng.*, **28**, 2839-2853 (1989).

[56] Levine, N., Stress sampling points for linear triangles in the finite element method, *Numer. Anal.*, **5**, 407-427 (1985).

[57] Ainsworth, M., Zhu, J.Z., and Craig ,A.W. and Zienkiewicz, O.C., Analysis of the Zienkiewicz-Zhu a-posteriori error estimator in the finite element method, *Int. J. Numer. Methods Eng.*, **28**, 2161-2174 (1989).

[58] Babuska, I., Planck, L. and Rodriguez, R., Basic problems of a posteriori error estimation, *Comput. Meths. Appl. Mech. Eng.*, **101**, 97-112 (1992).

[59] Carey, G.F., Chow, S.S. and Seager, M.K., Approximate boundary-flux calculations, *Comput. Meths. Appl. Mech. Eng.*, **50**, 107-120 (1985).

[60] Mizukami, A., A mixed finite element method for boundary flux computation, *Comput. Meths. Appl. Mech. Eng.*, **57**, 239-243 (1986).

[61] Fish, J., Markolefas, S., Guttal, R. and Nayak, P., On adaptive multilevel superposition of finite element meshes for linear elastotastics, *Applied Numerical Mathematics*, **14**, 135-164 (1994).

[62] Babuska, I. and Rodriguez, R., The problem of the selection of an a posteriori error indicator based on smoothening techniques, *Int. J. Numer. Methods Eng.*, **36**, 539-567 (1993).

[63] Misra, H. and Parsons, I.D., Adaptive finite element multigrid methods on parallel computers, in: B.H.V. Topping and M. Papadrakakis, eds., *Advances in Parallel and Vector Processing for Structural Engineering*, Civil-Comp Ltd, Edinburgh, Scotland (1994).

[64] Parsons, I.D. and Hall, J.F., The multigrid method in solid mechanis: part I - algorithms description and behaviour, *Int. J. Numer. Methods Eng.*, **29**, 719-738 (1990).

[65] Parsons, I.D. and Hall, J.F., The multigrid method in solid mechanis: part II - practical applications, *Int. J.Numer.Methods Eng.*, **29**, 739-754 (1990).

[66] Hackbush, W., *Multi-Grid Methods and Applications*, Springer-Verlag, Berlin (1985).

[67] McCormick, S.F., *Multigrid Methods*, SIAM, Philadelphia (1987).

[68] Bank, R.H., Dupont, T., and Yserentant, H., The hierarchical basis multigrid method, *Numerische Mathematik*, **52**, 427-458 (1988).

[69] Bank, R.E., and Yserentant, H., Some remarks on the hierarchical basis multigrid method, in: T.F. Chan, R. Glowinski, J. Periaux and O.B. Widlund, eds., *Proc. of the Second Int. Symposium on Domain Decomposition Methods*, SIAM, Philadelphia, 140-146, (1989).

[70] Yserentant, H., On the multi-level splitting of finite element spaces for indefinite elliptic boundary value problems, *SIAM J. Num. Anal.*, **23**, 581-595 (1986).

[71] Yserentant, H., Hierarchical bases give conjugate gradient type methods a multigrid speed of convergence, *Applied Mathematics and Computation*, **19**, 347-358 (1986).

[72] Coutinho, A.L.G.A., Alves, J.L.D., Ebecken, N.F.F. and Devloo, P.R., Two-level hierarchical preconditioners for finite element equations, in: I.D. Parsons and B. Nour-Omid, eds., *ASME Winter Annual Meeting*, Atlanta, 47-55 (Dec. 1991).

[73] Barra, L.P.S., Coutinho, A.L.G.A., Telles, J.C.F. and Mansur, W.J., Multi-level hierarchical preconditioners for boundary element systems, *Eng. Anal. Bound. Elem.*, **12**, 103-109 (1993).

[74] Hestenes, M.R., and Stiefel, E., Methods of conjugate gradients for solving linear systems, *J. Res. Nat. Bur. Stand.*, **49**, 409-436 (1952).

[75] Reid, J.K., On the method of conjugate gradients for the solution of large sparse systems of linear equations, in J.K. Reid, ed., *Large Sparse Sets of Linear Equations*, Academic Press, New York, 231-254 (1970).

[76] Brandt, A., Multi-level adaptative adaptive solutions to boundary-value problems, *Math. Comp.*, **31**, 333-390 (1977).

[77] Jouglard, C.E., and Coutinho, A.L.G.A., A comparison of iterative multi-level finite element solvers, *Comp.& Struc.*, submitted for publication (1996).

[78] Papadrakakis, M., and Bibilis, G.P., Solution techniques for the p-version of the adaptive finite element method, *Int. J. Numer. Methods Eng.*, **37**, 1413-1431 (1994).

[79] Silva, R.S., Garcia, E.L.M., and Coutinho A.L.G.A., An algorithm to control the algebraic error in adaptive refinement (in Portuguese), in *Proc. of the XII Brazilian Congress of Mechanical Engineering*, vol I, Brasilia, Brasil, 181-184 (1993).

[80] Mavriplis, D.J., Adaptive mesh generation for viscous flows using delauney triangulation, *J. Comp. Phys.*, **90 (2)**, 271-291 (1990).

[81] Mavriplis, D.J., Unstructured mesh generation and adaptivity, ICASE report no.95-26, NASA CR195069, April 1995.

[82] Parsons, I.D., Parallel adaptive multigrid methods for elasticity, plasticity and eigenvalue problems, in: M. Papadrakakis, ed., *Solving Large-scale Problems in Mechanics - Parallel and Distributed Computer Applications*, vol. II, John Wiley & Sons Ltd, Chichester, England (1996).

[83] Bout, A., A displacement-based geometrically non-linear constant stress element, *Int. J. Numer. Methods Eng.*, **36**, 1161-1188 (1993).

[84] Durlofsky, L.J., Accuracy of mixed and control volume finite element approximations to Darcy velocity and related quatities, *Water Resour. Res.*, **30 (4)**, 965-973 (1994).

[85] Luo, H., Baum, J.D., Löhner, R. and Cabello, J., Adaptive edge-based finite element schemes for the Euler and Navier-Stokes equations on unstructured grids, *31st Aerospace Sciences meeting & Exhibit*, Reno, NV, USA, AIAA - 93-0336. (1993).

A Multilevel Adaptive Approach for the Solution of Non-Linear Finite Element Problems

by

Claudio Ernesto Jouglard

Department of Civil Engineering
COPPE/UFRJ

Research Report EC-ES001/96

Contents

Chapter 1

Introduction

1.1 Background

Adaptivity is a term introduced in the early 1980s to describe an automated convergence process which attempts to distribute the basic approximation parameters of the finite element method in some optimal way. With the finite element method, the most commonly used adaptive strategies are classified as h and p refinement. In the former case the desired accuracy is achieved by decreasing the size h of the elements, whilst in the latter case by increasing the order p of the shape functions.

Implicit in the adaptive methods is the ability to somehow asses the 'quality' of an approximate solution locally over various elements in a given mesh. This quality of a numerical approximation is naturally judged by its error, that is, the difference between the exact solution u and its finite element approximation u_h, which is measured in some appropriate norm. Thus, an integral part of a rational adaptive scheme is some means to estimate *a-posteriori* (after one approximate solution is obtained for an initial mesh and element family) the local approximation error. The basic ideas of *a-posteriori* error estimation have been introduced by Babuska and Rheinboldt [14,15] and since then many error estimators appeared in the literature [16-33].

All the numerical methods requires at some stage the solution of a linear system of equations. Although direct solvers are enjoying great popularity in dealing with relatively small-scale systems of equations that arise from the conventional finite element method, in an adaptive formulation they suffer from excessive fill-in and computing time. In connection with adaptive analysis iterative solvers are preferable since a very good approximation to the solution is known from the previous coarse mesh analysis and they require a minimum storage.

In this work we analyze h-adaptivity in the context of two-dimensional problems of linear and non-linear elasticity. We present an error estimator for linear triangles and several iterative solvers employed in the adaptive analysis are compared. Finally the combination of iterative solvers and adaptivity is employed to solve some *geometrical* non-linear problems.

1.2 Thesis outline

In chapter 2, we present an error estimator for linear triangles based on stress projections. We use local side coordinates to define normal and tangential fluxes on each side, which are averaged between adjacent elements. At loaded boundaries the smoothed gradient field can be enforced to adopt the averaged values of prescribed fluxes in a simple and natural manner. Numerical comparisons are made of this error estimator with others of the same class.

In chapter 3 the data structure for the adaptive code was presented. It is organized by levels and consists of the traditional finite element data structure supplemented by additional arrays of connectivity information.

In chapter 4 the iterative multivel solvers are presented. Also a convergence criteria based in energy norms is presented and its relation with adaptive refinements is numerically analyzed. It must be noted that the energy criteria has the advantage of being non-dimensional.

In chapter 5 numerical comparisons were made for the iterative multilevel solvers in linear adaptive problems. Also in this chapter, some examples of geometrical non-linear problems with adaptivity were tested, to show the combined efficiency of the multilevel solvers and the adaptive strategy when applied to this type of problems.

Finally, in chapter 6 we summarize the results of this work and present fruitful areas for future research.

Chapter 2

Adaptive Mesh Refinement

2.1 Introduction

When using an approximate numerical method, such as the *finite element method*, to solve a mathematical model of a physical problem we have three type of errors: modelization errors due to simplifications assumed in the mathematical model, discretization errors due to the approximation of the continuous model by a discrete one and numerical errors due to approximate computations on the discrete model.

By computational analysis, only mathematical problems and not reality can be analized. The minimization of errors incurred during the formulation of the mathematical model is not considered in general, but this is an area of current active research and future developments will be expected, see for example refs. [9], [10].

The numerical error is due to the approximate solution of the system of equations arising from the discretization. It can be influenced by several factors such as the characteristics of the discrete model, parameters of the numerical methods, the limited precision of computers, etc. Basically all the numerical methods requires at some stage the solution of a linear system of equations, and a robust solver is essential to minimize this type of error.

Finally, we have the discretization error which arises from the approximation of the continuous mathematical model by a discrete model. A discretization is characterized by the finite element mesh and the choice of elements. The objective of an adaptive analysis is to control the discretization error by increasing the number of degrees of freedom in regions where the previous finite element model is not adequate and to achieve the specified accuracy in a most economic manner. It is therefore essential to have a quantitative assessment of the quality of the approximate solution and the capabilities of refinement.

An adaptive method must have two major components:

- *Error estimation*: This includes the methods and algorithms used for the estimation of the discretization error in approximate solutions. A robust error estimation scheme should give good indication of the error in coarse meshes and must provide an asymptotically exact estimate of the error, that is the

estimate should converge to the true error as the degrees of freedom of the mesh are adaptively incremented.

- *Mesh optimization*: This refers to the methods which are used to predict the optimal distribution of the degrees of freedom of the mesh based on the estimated error and the procedures employed to control the modifications of the mesh to achieve such optimal distribution.

In this chapter the mesh refinement strategy adopted is presented. First we introduce the definition of the discretization error. Then the refinement strategy based on this error is presented. Finally the error estimates used in this study are presented.

2.2 Discretization Error and Error Measures

A wide range of boundary value problems including all problems of linear elasticity are characterized by the following linear differential equation

$$\boldsymbol{L}\,\boldsymbol{u} + \boldsymbol{p} = \boldsymbol{0} \tag{2.1}$$

in the domain Ω with boundary conditions

$$\boldsymbol{u} = \bar{\boldsymbol{u}} \quad \text{on } \Gamma_u \tag{2.2.a}$$

$$\boldsymbol{M}\,\boldsymbol{u} = \bar{\boldsymbol{t}} \quad \text{on } \Gamma_t \tag{2.2.b}$$

where $\boldsymbol{L}$ and $\boldsymbol{M}$ are linear differential operators, and Γ_u, Γ_t are the portions of the boundary where $\boldsymbol{u}$ or some combination of its derivatives are prescribed, respectively.

A finite element approximation $\boldsymbol{u}_h$ to the solution $\boldsymbol{u}$ of this problem can be expressed as

$$\boldsymbol{u} \approx \boldsymbol{u}_h = \boldsymbol{N}\,\hat{\boldsymbol{u}} \tag{2.3}$$

where $\hat{\boldsymbol{u}}$ is the vector of nodal values and N is the matrix of shape functions. The *discretization error* is defined as the difference between the exact and finite element solutions, i.e.

$$\boldsymbol{e} = \boldsymbol{u} - \boldsymbol{u}_h \tag{2.4}$$

The specification of a local error in this manner is generally not convenient to identify the overall quality of the solution. For this reason various 'norms' representing some integral scalar quantity are often introduced to measure the error. A common measure is the 'energy norm'. The error in the energy norm is defined as

$$\|\boldsymbol{e}\| = \left(\int_\Omega \boldsymbol{e}^{\mathrm{T}}\mathbf{L}\,\boldsymbol{e}\,\mathrm{d}\Omega\right)^{1/2} \equiv \left(\int_\Omega (\boldsymbol{u}-\boldsymbol{u}_h)^{\mathrm{T}}\mathbf{L}\,(\boldsymbol{u}-\boldsymbol{u}_h)\,\mathrm{d}\Omega\right)^{1/2} \tag{2.5}$$

and when the differential operator $\mathbf{L}$ is self-adjoint, we have

$$\|\boldsymbol{e}\|^2 = \|\boldsymbol{u}\|^2 - \|\boldsymbol{u}_h\|^2 \tag{2.6}$$

It must be noted that this equation is only valid for discrete solutions $\boldsymbol{u}_h$ interpolated by conforming shape functions [12,13].

For linear plane elasticity problems the governing equilibrium equations and boundary conditions can be written as

$$\boldsymbol{S}^{T}\sigma + \boldsymbol{b} = \boldsymbol{0} \tag{2.7.a}$$

$$\boldsymbol{n}^{T}\sigma = \bar{t} \quad \text{on } \Gamma_t \tag{2.7.b}$$

$$\boldsymbol{u} = \bar{\boldsymbol{u}} \quad \text{on } \Gamma_u \tag{2.7.c}$$

where $\boldsymbol{S}$ is the first order strain differential operator which defines the strains as

$$\varepsilon = \boldsymbol{S}\,\boldsymbol{u} \tag{2.8}$$

and σ represents the stress vector which can be expresed as

$$\sigma = \boldsymbol{D}\varepsilon = \boldsymbol{D}\boldsymbol{S}\,\boldsymbol{u} \tag{2.9}$$

where $\boldsymbol{D}$ is the matrix of elastic constants.

For elasticity problems the energy norm of the displacements $\boldsymbol{u}$ is defined by Eqn.(2.5) and yields, on integration by parts

$$\|\boldsymbol{u}\| = \left(\int_{\Omega} (\mathbf{S}\boldsymbol{u})^{T}\,\mathbf{D}\,(\mathbf{S}\boldsymbol{u})\,d\Omega\right)^{1/2} \tag{2.10}$$

and the error in the energy norm is

$$\|\boldsymbol{e}\| = \left(\int_{\Omega} (\mathbf{S}\boldsymbol{e})^{T}\,\mathbf{D}\,(\mathbf{S}\boldsymbol{e})\,d\Omega\right)^{1/2} \tag{2.11}$$

The approximate displacements $\boldsymbol{u}_h$ gives an approximate stress field σ_h as

$$\sigma_h = \boldsymbol{D}\boldsymbol{S}\,\boldsymbol{u}_h \tag{2.12}$$

then the error in the energy norm can be written as

$$\|\boldsymbol{e}\| = \left(\int_{\Omega} (\sigma - \sigma_h)^{T}\,\mathbf{D}^{-1}\,(\sigma - \sigma_h)\,d\Omega\right)^{1/2} \tag{2.13}$$

It must be noted that the error in the energy norm can be evaluated by summing over all the elements as

$$\|\boldsymbol{e}\|^2 = \sum_{i=1}^{ne} (\|\boldsymbol{e}\|_i)^2 \tag{2.14}$$

where i refers to individual elements Ω_i such that their union is the complete domain Ω.

The relative error in the energy norm is defined as

$$\eta = \frac{\|\boldsymbol{e}\|}{\|\boldsymbol{u}\|} \tag{2.15}$$

A specified maximum value $\bar{\eta}$ is generally used as the criterion for the adaptive analysis process.

2.3 Adaptive Refinement Strategies

Several types of adaptive refinement are identified. The most common one is the *h*-type, which achieves the accuracy by refining the mesh using a given type of finite element. The *p*-type keeps the mesh fixed and the accuracy is achieved by increasing the order of the approximation used. The *h-p* type is the proper combination of *h*- and *p*-types. Other types of adaptive refinement exists, but are less used, such as the *r*-method where the nodes of the mesh are relocated to satisfy a set of optimal conditions.

For the *h*-version of the finite element method *a priori* error estimates have been developed for a wide range of finite elements. In particular for linear elliptic problems if the mesh is uniformly refined with the size of the element, h, tending to zero while the polynomial degree p of the approximation is fixed the following estimate [12] can be established

$$\|e\| \leq C\, h^{\min(p,\lambda)} \approx C\, N^{-1/2 \min(p,\lambda)} \tag{2.16}$$

where $\|e\|$ is the error in the energy norm, C is a positive constant independent of h and p, λ is a positive constant depending on the strength of the singularity present (λ = 0.4-0.8 typically) and N is the number of degrees of freedom, which in a given problem in two dimensions is approximately proportional to h^{-2}. Here h is the maximum distance between two points in an element, which in the case of triangular elements coincides with the length of its largest side.

The construction of an optimal mesh is generally based on the principle of equidistribution of error among elements [27,44,45]. Following this principle the error must be approximately equal on each element. Taking into account that for energy norms only the square of these norms are additive, then for an optimal mesh of ne elements we must have for each element i

$$\|e\|_i \approx \|e\| / \sqrt{ne} \tag{2.17}$$

Also it is found that for an optimal mesh the influence of the singularities is eliminated [46], giving

$$\|e\| \leq C\, h^p \approx C\, N^{-p/2} \tag{2.18}$$

and optimal convergence rate is achieved. One of the main tasks of *h*-version adaptive analysis is to achieve this higher rate of convergence, and to accomplish this the use of a nearly optimal mesh is imperative.

The refinement strategy employed to construct a quasi-optimal mesh will depend on the nature of the criteria on accuracy which we wish to satisfy. A very common requirement is to specify the achievement of a certain minimum relative error in the energy norm. We thus require that after the final analysis is completed the condition

$$\eta = \frac{\|e\|}{\|u\|} \leq \overline{\eta} \tag{2.19}$$

be satisfied for the whole domain, where $\overline{\eta}$ is the maximum permissible error.

Assuming that the error is equally distributed between elements this requirement can be written for each element using the Eqn. (2.17) as

$$\|e\|_i \leq \overline{\eta}\,\|u\|/\sqrt{ne} = e_m \tag{2.20}$$

where e_m is the *mean error.*

Then we can define for each element a refinament ratio ξ_i as

$$\xi_i = \frac{\|e\|_i}{e_m} \tag{2.21}$$

which defines the elements to be refined when $\xi_i > 1$.

To construct an optimal mesh we need a *mesh-size function h(x,y)* which predicts the optimal mesh-size h_{new} for each element. If the current element size is h_{old}, and we assume the rate of convergence of the error to be optimal $O(h^p)$ (see Eqn.(2.18) then the predicted element size should be

$$h_{new} = h_{old} / (\xi_i)^{1/p} \quad , \qquad \text{for } \xi_i > 1 \tag{2.22}$$

This expression can be used too when $\xi_i < 1$ indicating that in some areas a coarser subdivision is permissible.

2.3.1 Refinement and Remeshing

There are basically two techniques to obtain the desired element size distribution namely *mesh regeneration* and *mesh refinement.* Using mesh regeneration a *remeshing* algorithm tries to generate the final grid in one pass. Different implementations have appeared in the open literature [47-50] based on the *advancing front* method introduced by Lo [47]. The grid is progressively generated as an *advancing generation front* is sweeping through the domain. Peraire et al. [48] extended Lo's algorithm to generate adaptive grids incorporating a mesh size function. Another type of remeshing techniques are based on Delaunay triangulations [79] and a recent review on different techniques of remeshing are discussed in detail in [80], where an extensive bibliography about the subject is given.

Using adaptive mesh refinement, triangular or quadrilateral elements are subdivided to form a group of four new elements as shown in Fig. 2.1. When an element is subdivided, irregular connections may result between the new elements and the neighbors of the original element. As we will see later, these irregular connections require special treatment in order to mantain the continuity of the approximation across the interelement boundaries. To construct a quasi-optimal mesh, several *refinement cycles* are employed on a given initial coarse grid. During each refinement cycle, the algorithm computes the refinement ratios ξ_i and loops through the elements of the mesh subdividing once every element with $\xi_i > 1$. It must be noted that the mean error definition as given by Eqn. 2.20 is not adequate to be used with mesh refinement, since it is intended to give the final

element size in one remeshing cycle. When used in each refinement cycle gives excessive refinements of coarse elements. A better criterion to be used with mesh refinement is to equidistribute the current error of the mesh defining the mean error as:

$$e_m = \|e\| / \sqrt{ne} \tag{2.23}$$

The use of this definition of mean error in conjunction with mesh refinement gives meshes with a better distribution of errors and with less elements, but in general more refinement cycles are necessary.

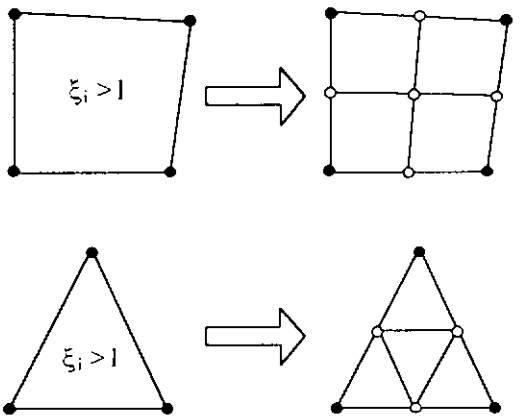


Figure 2.1. Regular subdivision of quadrilaterals and triangles.

As noted by Strouboulis and Haque [36] *h-remeshing* appears to be more efficient than *h-refinement* since it can produce solutions of given accuracy with fewer number of degrees of freedom, but they pointed out that the computer time required to construct a new mesh using remeshing can be very large in comparison with the time required to construct a comparable mesh using refinement.

Here we will adopt the *h-refinement* technique since it has several advantages over *h-remeshing*, for example

- It is more adequate for multilevel solvers which need a hierarchy of refined meshes.
- Simple propagation of state variables from one level of refinement to another.
- Simple coupling of multiple solutions when the same coarse mesh is used. This is important, for example, in modal dynamic analysis and bifurcation problems.
- It is more appropriate for evolution problems where few changes of the optimal mesh occurs between steps of the analysis, such as in wave propagation problems [8].

In particular we implement the linear triangle in an adaptive code for linear and non-linear elasticity, and the details of implementation are referred to this element.

2.3.2 Mesh Refinement and Transitions

In this study we adopt a regular refinement criteria for triangles where each element is subdivided into four similar ones (Fig. 2.1). The successive application of the subdivision process leads to a natural arrangement of elements by levels. All the elements in the coarsest mesh have level zero. When a particular element is refined, the four new elements have a level one unit higher than their parent element. Similarly, every side and node is assigned a level, that correspond to the first appearance of them in the mesh (Fig. 2.2).

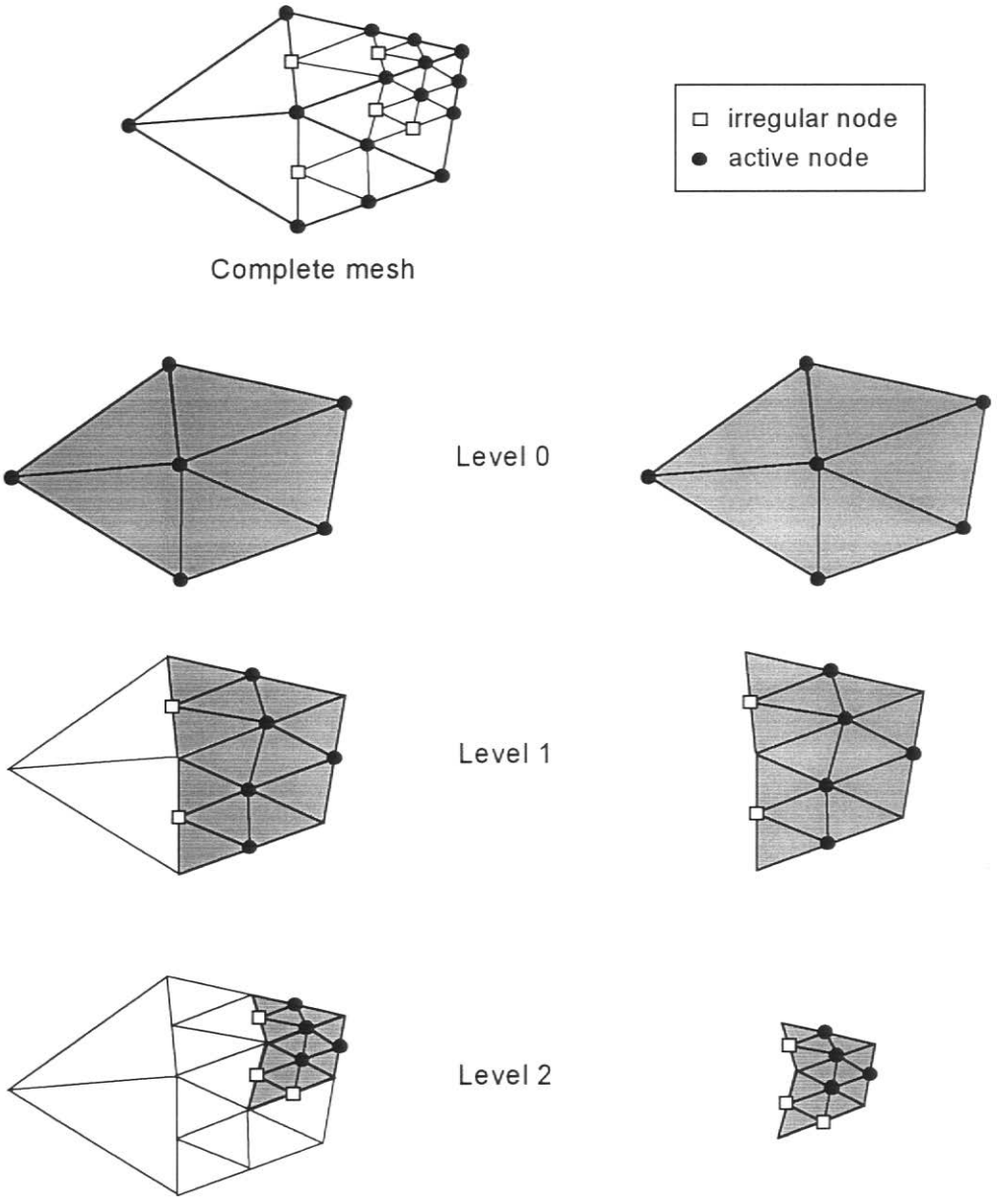


Figure 2.2. Level partitioning of the mesh.

As was previously mentioned, when an element is subdivided, irregular connections may result between the new elements and the neighbors of the original element. In order

to obtain a smooth transition in size we impose the restriction that any element must have only up to one irregular connection on one of its sides. This is equivalent to say that levels of adjacent elements must be equal or consecutive. This rule is known as the *1-irregular rule* [36].

In order to attend the *1-irregular rule* several procedures can be applied. Here we adopt the next. First we separate the elements of each level in two types: *refinable* and *unrefinable*. An element will be *unrefinable* if it has an adjacent element of an inferior level and it can not be subdivided. Note that, if exist, *unrefinable* elements are located at the boundaries of each level. All elements that are not *unrefinable* are considered *refinable*. Then we loop over all the levels of the mesh in ascending order subdividing only the *refinable* elements that must be refined. Before the refinement of each level we reclassify their elements if the number of *refinable* and *unrefinable* elements have changed due to refinement of the previous level. Using this procedure the elements of each level can be subdivided only once in each refinement cycle.

Due to the local refinement each level has refined and unrefined elements. At the boundary between refined and not refined elements we have *irregular nodes*, that is nodes that have no associated interpolation functions on some adjacent triangles. An element containing an *irregular* node will be called a *transition element* (Fig. 2.3), and the sides of these elements containing an *irregular* node will be called *transition sides*. It must be noted that with the restrictions imposed by the *1-irregular rule* transition sides can be subdivided only once since these sides are on the boundary between adjacent levels.

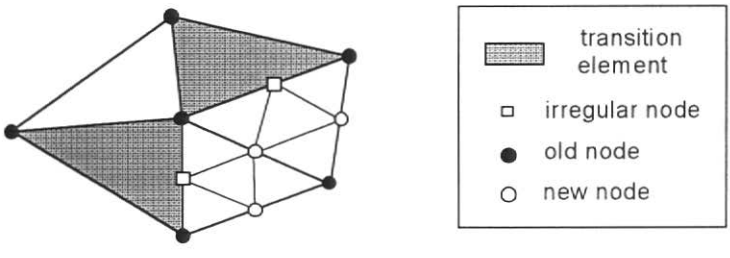


Figure 2.3. Transition elements and irregular nodes.

It must be noted that the *1-irregular* rule gives locally *structured* meshes in each refined level, that is, the number of elements adjacent to each node of these levels is constant. We can see in Fig 2.2 that six elements are attached to an interior node of each refined level, and three elements are attached to both irregular and boundary nodes of a refined level. This property can be useful for implementations on vector and parallel computers.

Irregular nodes must be treated adequately in order to maintain the consistency of the mesh. There are several approaches based on the subdivision of the transition elements. For triangles the standard approach [3] consists in the subdivision of the transition

element by a new side joining the intermediate irregular node with the opposite vertex of the triangle (Fig 2.4).

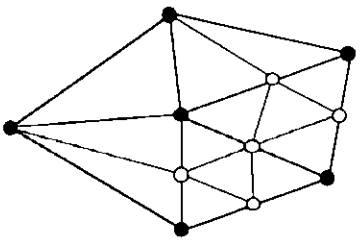


Figure 2.4. Triangular refinement - Standard approach.

It must be noted that for the implementation of this procedure we must refine previously all the transition elements having more than one transition side, see for example [8].

Another method is suggested by Rivara [4], which consists in joining the irregular node with the opposite vertex, if this irregular node is located on the longest side of the transition element, else it is joined with the mid-side point of the longest side of the transition triangle, where a new node is generated (Fig. 2.5). The process finish when all the irregular nodes are eliminated. This second approach leads to a moderately wider spreading of the refinement zone and has the advantage that the distortion of the elements generated over the successive refinements can not be greater than the original mesh. But this procedure gives *unstructured* meshes in each level.

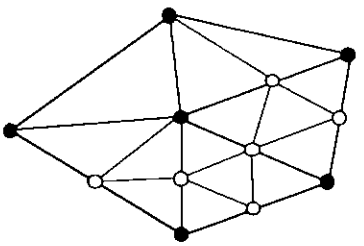


Figure 2.5. Triangular refinement - Rivara's method.

In order to assure continuity of the shape functions over the domain and to retain local structured meshes in each level we adopt another approach, usually employed with quadrilateral elements [6,7,51], that does not need subdivision of transition elements. For linear elements, we will constrain the value of the functions at the irregular node to be the average of the values at the end nodes of the transition side (Fig. 2.6).

The use of constrained nodes has the advantage that since all the elements are subdivided in the same manner the logic involved is simpler than that required for the other alternatives. On the other hand, the use of constraints requires the application of suitable matrix transformations to compute matrix-vector products and dot products, when matrices and vectors are expressed in the unconstrained system.

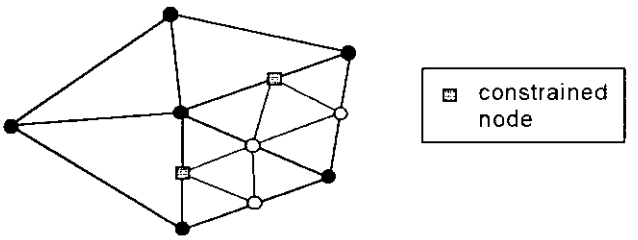


Figure 2.6. Regular refinement with constrained nodes.

It must be noted that if the optimal mesh changes during the computations, such as in evolution and non-linear problems, is convenient to extent the refinement region to anticipate these changes. Here we adopt the procedure suggested by Löhner [8], where several 'protective' layers of elements are added to surround the elements to be refined. To identify the elements of each layer we loop over the elements to be refined marking all nodes connected to these elements. Then we loop over all the elements, if at least one node of a given element has been marked this element must be refined. The process is repeated for each additional protective layer.

In the current implementation at least one protective layer is added to each refined region, even for linear problems, in order to translate the *unrefinable* elements from the boundaries of the refined region to the 'protective' layers. Also, to simplify the logic involved, we have limited the protective layers to be composed of elements of the same level that is being refined.

2.4 Error Estimators for Adaptive Analysis

Since the exact error of a current discrete solution is not known in advance, the efficient implementation of the mesh optimization procedures driven by error measures needs robust estimators of these measures. In the next sections we describe the estimators implemented in this study.

2.4.1 *A Posteriori* Error Estimation

The *a posteriori* error estimates are based on information obtained during the solution process. In this respect, they are different from the theoretical *a priori* error estimates which predict the rate of convergence but say little about the error in a particular solution on a finite element mesh. The basic ideas have been introduced in [14,15] and since then many *a posteriori* error estimators appeared in the literature.

These estimators are essentially of the following types:

(a) *Residual estimators.* These employ the residuals of the finite element solution. For more about these we refer among others to [14-25]. These estimators often are based on complementary energy principles [16,17,22,23].

(b) *Flux-projection operators.* These estimators are based on the idea of smoothing the fluxes (which are discontinuos) and comparison with the fluxes computed from the original finite element solution [29-33].

(c) Other categories include extrapolation estimators [26] and interpolation-error bounds [27,28].

In [34-36] the comparison of the performance of various major error estimators and their robustness is given.

Here we adopt *flux-projection operators* for the construction of the error estimators. These type of error estimators are based on the use of post-processing techniques which recover more accurate values of derivatives from the original finite element solution. For elasticity problems the recovered stress fields, which are expected to be more precise than those obtained by direct derivation of the finite element displacement fields, can be used to approximate the exact stress fields.

Then we can replace the exact stress σ by the recovered stress σ^* in Eqn. (2.13) to obtain an error estimator $\|e^*\|$ in the energy norm as

$$\|e^*\| = \left(\int_\Omega (\sigma^* - \sigma_h)\, \mathbf{D}^{-1}\, (\sigma^* - \sigma_h)\, d\Omega\right)^{1/2} \tag{2.24}$$

and assuming that conforming finite elements are employed, an estimate $\|u^*\|$ of the energy norm of the exact solution can be obtained using Eqn. (2.6) as

$$\|u^*\|^2 = \|u_h\|^2 + \|e^*\|^2 \tag{2.25}$$

If non-conforming finite elements are employed, we simply estimate $\|u^*\| = \|u_h\|$, which is a satisfactory approximation specially if the error is small [54].

Then a relative error estimator η^* can be computed as

$$\eta^* = \frac{\|e^*\|}{\|u^*\|} \tag{2.26}$$

Several procedures exist [30,32,33,37-40] for the computation of the smoothed stress field, and some of them will be described in the next section.

The effectivity of the error estimators is measured by an *effectivity index* θ given by

$$\theta = \frac{\|e^*\|}{\|e\|} \tag{2.27}$$

that is, the ratio of the norms of the estimated to the true error. The error estimator is said to be *asymptotically exact* if θ converges to unity whenever the true error converges to zero with decreasing mesh size *h*.

2.4.2 General Form of Smoothed Stress Fields

The finite element stress σ_h computed directly from the finite element solution by Eqn. (2.12) is generally discontinuos across element interfaces. In general it does not satisfy the equilibrium conditions neither in the interior nor on the boundaries of the domain. It is reliable only at the global level, in certain mean sense.

In many circumstances, such as in graphical representation of finite element stress fields and in the estimation of the discretization error, it is interesting to obtain a continuos stress field. This can be achieved in two steps.

First, we must estimate the stresses at certain selected points. They are generally chosen as nodes associated with shape functions of the finite element model. This allows to define $\hat{\sigma}^*$, the vector of *estimated nodal stresses.*

Secondly, by using a standard finite element interpolation, estimated stresses can be evaluated at any point by the following formula

$$\sigma_h = \tilde{N}\,\hat{\sigma}^* \tag{2.28}$$

where $\tilde{N}$ is the stress smoothing matrix, which contains standard shape functions.

The quality of the smoothed stress field to approximate the exact stress field depends, among other factors on the mesh density. That is, a very coarse mesh can give a smoothed stress field with highly under or over estimated stresses, this is generally indicated by severe interelement discontinuities of the unaveraged stresses. Continuity of the stresses must not be imposed at the interfaces between different materials or between elements with different thickness.

2.4.3 Simple Nodal Averaging

An easy way of obtaining a smoothed stress field for linear elements consists in averaging nodal stresses from the elements connected at each node

$$\hat{\sigma}^* = \left(\sum_{i=1}^{n} \sigma_{hi} \right) / n \tag{2.29}$$

where n is the number of elements surrounding the node.

The recovered nodal values are influenced by the size of the elements attached to the node and better results are obtained when these elements have similar sizes, specially in regions of high gradients. But if the mesh is adaptively designed abrupt changes of sizes between elements are limited.

2.4.4 Superconvergent Patch Recovery

Numerical experiments show that for some classes of problems nodal displacements are more accurate, sometimes even exact, that in other locations of the mesh. This property is often designated as the nodal *superconvergence* of displacements [41]. Based on this property and by using Taylor series expansion, superconvergent points of stresses can be found inside, or at the boundary, of each element [41,42].

Zienkiewicz and Zhu [30,31] have proposed a stress smoothing method based on a superconvergent patch recovery of stresses which is also known as the SPR procedure. In this technique the smoothed nodal stresses are assumed to belong to a polynomial expansion of the same complete order as that present in the shape functions of the element, and it is assumed that this expansion is valid over an element patch surrounding the particular assembly node considered. Such a *patch* represents an union of elements containing this vertex node (in a manner similar to that used in the *patch test* [11-13]). This polynomial expansion can be written as

$$\sigma^* = \mathbf{P}\,\mathbf{a} \tag{2.30}$$

where **P** contains the appropiate polynomial terms and **a** is a set of unknown parameters. For two dimensions and linear expansion we have

$$\mathbf{P} = \{1 \;\; x \;\; y\} \tag{2.31}$$

and

$$\mathbf{a} = \{a_1 \;\; a_2 \;\; a_3\}^T \tag{2.32}$$

The determination of the unknown parameters **a** is best made by ensuring a least square fit of this to the set of superconvergent or at least high accuracy sampling points existing in the patch considered. To do this we minimize

$$F(\mathbf{a}) = \sum_{i=1}^{n} [\sigma_h(x_i,y_i) - \sigma^*(x_i,y_i)]^2 = \sum_{i=1}^{n} [\sigma_h(x_i,y_i) - \mathbf{P}(x_i,y_i)\mathbf{a}]^2 \tag{2.33}$$

where (x_i, y_i) are the coordinates of the n sampling points of the patch.

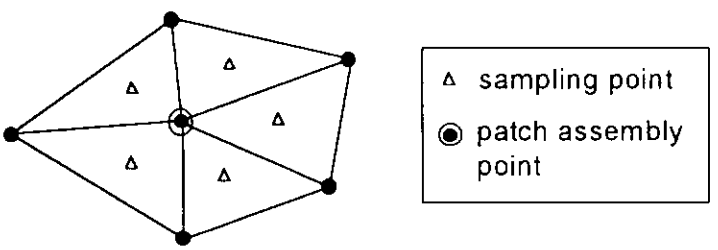


Figure 2.7. Computation of nodal superconvergent values for linear triangles.

The minimization condition of F(**a**) implies that **a** satisfies

$$\sum_{i=1}^{n} [\mathbf{P}^T(x_i,y_i)\,\mathbf{P}(x_i,y_i)]\,\mathbf{a} = \sum_{i=1}^{n} \mathbf{P}^T(x_i,y_i)\,\sigma_h(x_i,y_i) \tag{2.34}$$

This can be solved in matrix form as

$$\mathbf{a} = \mathbf{A}^{-1}\,\mathbf{b} \tag{2.35}$$

where

$$\mathbf{A} = \sum_{i=1}^{n} [\mathbf{P}^{T}(x_i,y_i)\ \mathbf{P}(x_i,y_i)] \tag{2.36.a}$$

$$\mathbf{b} = \sum_{i=1}^{n} \mathbf{P}^{T}(x_i,y_i)\ \sigma_h(x_i,y_i) \tag{2.36.b}$$

The number of equations to be solved on each patch is modest and the same matrix **A** occurs in the solution for each component of σ^*.

Once the parameters **a** are determined the recovered nodal values $\hat{\sigma}^*$ are simply calculated by insertion of appropriate coordinates into the expression for σ^*.

For nodes located at the boundary or at interfaces between elements of different materials a special treatment is required [43]. In the implementation made here it is adopted the procedure of nodal averaging for the boundary nodes and the superconvergent patch recovery technique is applied only for interior nodes.

The method has been improved adding the squares of the residuals of the equilibrium equations and natural boundary conditions to the function F(**a**) in Eqn.(2.33) [32,33].

For linear triangles Zienckiewicz and Zhu [30] suggest to use the centroidal point as sampling point, though the assumption of superconvergence is not rigorously correct for this point. It must be noted that the intention is to use superconvergence points as sampling points to obtain a superconvergent interpolation, but for some elements and meshes these superconvergent points do not exist [42]. In general, the performance of the estimator depends on the selection of the sampling points, but it has been demonstrated that the robustness of the SPR procedure is rather insensitive to the selection of the sampling points [34].

2.4.5 Side Flux Averaging

We develop here an error estimator for linear triangles based on the linear interpolation of averaged values of fluxes at mid-side locations. We use local side coordinates to define normal and tangential fluxes on each side, so that at loaded boundaries the smoothed gradient field can be enforced to adopt the values of prescribed fluxes in a simple and natural manner. As it will be shown in the numerical examples, the consideration of the prescribed fluxes in the construction of the smoothed field results in an improved performance of the estimator.

In particular for stress analysis we use the total normal and tangential forces acting on each side as the averaging quantities since they must be equal for adjacent elements even if the stress field is discontinuous due to abrupt material or thickness changes. In this manner no distinction must be made between elements with different properties.

The use of sampling points at mid-sides of adjacent linear elements has the advantage that the averaged derivatives are superconvergent at this locations [55], and since any asymptotically exact estimator is closely related to superconvergence [57] a good performance of the estimator can be expected. But, on the other hand, a linear

interpolation of these mid-side values gives a discontinuous gradient field between adjacent elements. This in principle can be interpreted as a disadvantage if the purpose of the recovery process is the graphical representation of finite element stress fields, but it must be noted that the discontinuities in the plots of the recovered stresses serve as visual indicators of the quality of the solution, giving a valuable information that it lost when continuity is enforced.

For unstructured meshes this estimator can be efficiently implemented in vector and parallel codes if an adequate side oriented data structure is adopted.

2.4.5.1 Side Flux Averaging for Scalar Fields.

Consider for simplicity a second order problem (*Poisson's equation*)

$$-\Delta u = f \quad \text{in } \Omega \tag{2.37.a}$$

$$u = \overline{u} \quad \text{on } \Gamma_u \tag{2.37.b}$$

$$\frac{\partial u}{\partial n} = \overline{q} \quad \text{on } \Gamma_q \tag{2.37.c}$$

where Ω is a planar domain discretized by linear finite elements, Γ_u is the part of the boundary where the variable u has prescribed values and Γ_q is the part of the boundary where the normal derivative $\partial u/\partial n$ has prescribed values.

For each element we define positive directions for the normal and tangential axes of each side. In particular we adopt as positive the outward normal and the clockwise tangential directions as showed in Fig.2.8.

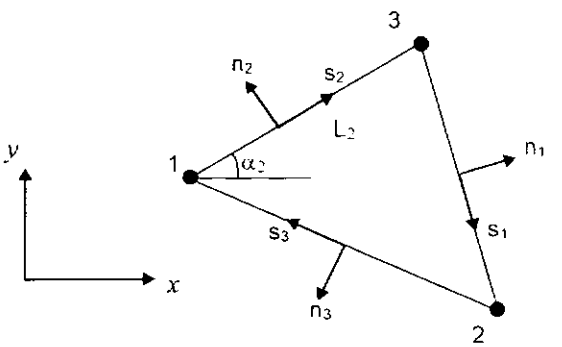


Figure 2.8. Tangential and normal axes of each side.

Since averaged first derivatives are assumed superconvergent at mid-side points then it is possible to obtain smoothed flux fields in a simple manner by averaging the mid-sides derivatives $\partial u_h/\partial x$, $\partial u_h/\partial y$ of the finite element approximation u_h between adjacent elements. Then interpolating these mid-side values with linear polynomials we obtain smoothed stresss fields. We denominate this procedure as *simple side averaging*.

Noting that natural boundary conditions are specified only in the direction of the normal to the boundary, then it is convenient to consider in each side the normal and tangential derivatives $\partial u/\partial n$, $\partial u/\partial s$ which can be expressed as

$$\frac{\partial u}{\partial n} = \cos\alpha \frac{\partial u}{\partial y} - \sin\alpha \frac{\partial u}{\partial x} = \boldsymbol{n} \cdot \nabla u \tag{2.38.a}$$

$$\frac{\partial u}{\partial s} = \sin\alpha \frac{\partial u}{\partial y} + \cos\alpha \frac{\partial u}{\partial x} = \boldsymbol{s} \cdot \nabla u \tag{2.38.b}$$

where α is the angle between the tangential axis of the side and the x axis, $\boldsymbol{n}$, $\boldsymbol{s}$ are the unit normal and tangential vectors to each side and ∇ is the gradient operator.

Then it is possible to obtain averaged values of the mid-sides normal derivatives $\partial u_h/\partial n$ of the finite element approximation u_h between adjacent elements. To do this we compute for each element e_i the normal derivative on its side L_i as

$$\left.\frac{\partial u_h}{\partial n}\right|_{L_i}^{e_i} = \cos\alpha \left.\frac{\partial u_h}{\partial y}\right|_{L_i}^{e_i} - \sin\alpha \left.\frac{\partial u_h}{\partial x}\right|_{L_i}^{e_i} = \boldsymbol{n}_i \cdot \nabla u_h^{e_i} \tag{2.39}$$

where it must be noted that for linear elements this normal derivative is constant along the side but has in general different values for adjacent elements.Then for each interior side L_i of the element e_1 we compute an averaged value of the mid-side normal derivative as

$$\left.\frac{\partial u^*}{\partial n}\right|_{L_1}^{e_1} = \frac{1}{2}\left(\left.\frac{\partial u_h}{\partial n}\right|_{L_i}^{e_1} - \left.\frac{\partial u_h}{\partial n}\right|_{L_i}^{e_2} \right) = \frac{1}{2} \boldsymbol{n}_i^{e_1} \cdot \left(\nabla u_h^{e_1} - \nabla u_h^{e_2} \right) \tag{2.40}$$

where e_2 is an adjacent element and the minus sign takes into account the fact that normal directions are opposite for adjacent elements.

If we have a prescribed normal flux $\partial u/\partial n = \bar{q}$ at a boundary side L_b of the element e_1 we impose a mean value of this flux to the mid-side normal derivative as

$$\left.\frac{\partial u^*}{\partial n}\right|_{L_b}^{e_1} = \int_{L_b} \bar{q}(s)\, ds \tag{2.41}$$

To define completely the averaged derivatives at mid-side points we must compute an averaged value of the tangential derivative. For each element e_i the tangential derivative on its side L_i is

$$\left.\frac{\partial u_h}{\partial s}\right|_{L_i}^{e_i} = \frac{u_{h2} - u_{h1}}{L} \tag{2.42}$$

where u_{h1}, u_{h2} are the values of the function u_h at the extremes of the side L_i whose length is L. It must be noted that these derivatives are identical for adjacent elements and are superconvergent at mid-side points (These can be deduced from the *mean value theorem*). Then there is no need to store an averaged value of the tangential derivative

between adjacent elements since it can be directly computed for each element from its nodal values as

$$\left.\frac{\partial u^*}{\partial s}\right|_{L_i}^{e_i} = \left.\frac{\partial u_h}{\partial s}\right|_{L_i}^{e_i} = \left.\frac{u_{h2} - u_{h1}}{L}\right|_{L_i}^{e_i} \tag{2.43}$$

This implies that only one averaged value per side must be stored. Note that if we were using *simple side averaging* we must store two averaged values per side.

With the averaged values for tangential and normal derivatives at mid-side points we can compute averaged values for the derivatives in the direction of the global axes as

$$\left.\frac{\partial u^*}{\partial x}\right|_{L_i}^{e_i} = \cos\alpha \left.\frac{\partial u^*}{\partial s}\right|_{L_i}^{e_i} - \sin\alpha \left.\frac{\partial u^*}{\partial n}\right|_{L_i}^{e_i} \tag{2.44.a}$$

$$\left.\frac{\partial u^*}{\partial y}\right|_{L_i}^{e_i} = \sin\alpha \left.\frac{\partial u^*}{\partial s}\right|_{L_i}^{e_i} + \cos\alpha \left.\frac{\partial u^*}{\partial n}\right|_{L_i}^{e_i} \tag{2.44.b}$$

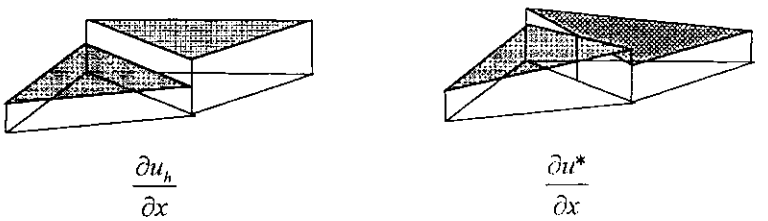
Finally we can interpolate the values of these derivatives over each triangle with linear shape functions

$$\frac{\partial u^*}{\partial x} = \sum_{i=1}^{3} M_i \left.\frac{\partial u^*}{\partial x}\right|_{L_i}^{e_i} \tag{2.45.a}$$

$$\frac{\partial u^*}{\partial y} = \sum_{i=1}^{3} M_i \left.\frac{\partial u^*}{\partial y}\right|_{L_i}^{e_i} \tag{2.45.b}$$

where the functions M_i are linear polynomials [13, pag.250] defined in terms of area coordinates as

$$M_i = 1 - 2\xi_i \tag{2.46}$$



$\frac{\partial u_h}{\partial x}$ $\frac{\partial u^*}{\partial x}$

Figure 2.9. Original and smoothed derivatives.

Then we can employ this smoothed field for the computation of error estimates. For this particular scalar elliptic problem the square of the error estimate in the energy norm (Eqn. 2.5) for each element can be written as

$$\|e^*\|_i^2 = \int_{\Omega_i} e^* \mathbf{L}\, e^* \, d\Omega_i \equiv \int_{\Omega_i} \left(\frac{\partial u^*}{\partial x} - \frac{\partial u_h}{\partial x} \right)^2 + \left(\frac{\partial u^*}{\partial y} - \frac{\partial u_h}{\partial y} \right)^2 d\Omega_i \qquad (2.47)$$

Although the smoothed field is only continuos at mid-side points (Fig. 2.9) it provides a good estimation for the errors as we will see in the examples.

2.4.5.2 Side Flux Averaging for Stress Fields.

For linear elasticity problems in two dimensions, we have two independent variables u, v which correspond to displacements in the x, y directions. The strains are defined as

$$\varepsilon_x = \frac{\partial u}{\partial x} \qquad (2.48.a)$$

$$\varepsilon_y = \frac{\partial v}{\partial y} \qquad (2.48.b)$$

$$\gamma_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \qquad (2.48.c)$$

and for plane stress conditions the stresses are

$$\sigma_x = \frac{E}{1-\nu^2}(\varepsilon_x + \nu\, \varepsilon_y) = \frac{E}{1-\nu^2}\left(\frac{\partial u}{\partial x} + \nu \frac{\partial v}{\partial y}\right) \qquad (2.49.a)$$

$$\sigma_y = \frac{E}{1-\nu^2}(\varepsilon_y + \nu\, \varepsilon_x) = \frac{E}{1-\nu^2}\left(\frac{\partial v}{\partial y} + \nu \frac{\partial u}{\partial x}\right) \qquad (2.49.b)$$

$$\tau_{xy} = \frac{E}{2(1+\nu)}\gamma_{xy} = \frac{E}{2(1+\nu)}\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) \qquad (2.49.c)$$

For plane strain conditions we substitute the elastic constants E, ν by $\hat{E} = E/(1-\nu^2)$ and $\hat{\nu} = \nu/(1-\nu)$.

It must be noted that if for two adjacent elements, having the same thickness and elastic constants, first derivatives of displacements are continuous and superconvergent at mid-side points then stresses must be continuous and superconvergent at these points too.

Then it is possible to obtain smoothed stress fields using *simple side averaging* to compute averaged values of the mid-side stresses σ_x^h, σ_y^h, τ_{xy}^h of the finite element approximate displacements u_h, v_h between adjacent elements. Interpolating these mid-side values with linear polynomials we obtain smoothed stress fields. But as in the scalar case is convenient to specify the mid-side stresses in normal and tangential directions, since boundary conditions on stresses are specified as prescribed tractions in these directions. Then rotating the stress tensor at each side the normal and tangential stresses σ_n, τ_{ns} can be obtained as

$$\sigma_n = \sigma_x (\sin\alpha)^2 + \sigma_y (\cos\alpha)^2 - 2\,\tau_{xy} \sin\alpha \cos\alpha \qquad (2.50.a)$$

$$\tau_{ns} = (\sigma_y - \sigma_x) \sin\alpha \cos\alpha + \tau_{xy} [(\cos\alpha)^2 - (\sin\alpha)^2] \qquad (2.50.b)$$

where α is the angle between the tangential axis of the side and the x axis, and we adopt the same convention as in the previous case to define the positive directions for the normal and tangential axes of each side which is shown in Fig.2.8.

For these local axes the strains can be written as

$$\varepsilon_s = \frac{\partial u_s}{\partial s} \qquad (2.51.a)$$

$$\varepsilon_n = \frac{\partial v_n}{\partial n} \qquad (2.51.b)$$

$$\gamma_{ns} = \frac{\partial u_s}{\partial n} + \frac{\partial v_n}{\partial s} \qquad (2.51.c)$$

where u_s, v_n are the displacements of the side in the tangential and normal directions, respectively. Then for these local axes the stress-strain relations can be written as

$$\sigma_s = \frac{E}{1-\nu^2} (\varepsilon_s + \nu\, \varepsilon_n) = \frac{E}{1-\nu^2} \left(\frac{\partial u_s}{\partial s} + \nu \frac{\partial v_n}{\partial n}\right) \qquad (2.52.a)$$

$$\sigma_n = \frac{E}{1-\nu^2} (\varepsilon_n + \nu\, \varepsilon_s) = \frac{E}{1-\nu^2} \left(\frac{\partial v_n}{\partial n} + \nu \frac{\partial u_s}{\partial s}\right) \qquad (2.52.b)$$

$$\tau_{ns} = \frac{E}{2(1+\nu)} \gamma_{ns} = \frac{E}{2(1+\nu)} \left(\frac{\partial u_s}{\partial n} + \frac{\partial v_n}{\partial s}\right) \qquad (2.52.c)$$

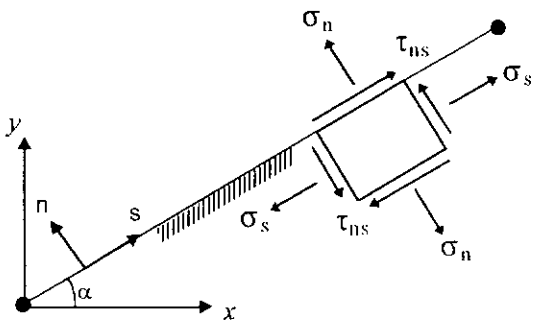


Figure 2.10. Tangential and normal stresses for local axes of each side.

Then we can compute averaged values at the mid-sides of the normal and tangential stresses σ_n^h, τ_{ns}^h of the finite element approximations u_h, v_h between adjacent elements. But at abrupt material property change interfaces stresses are not continuous, but the

normal and tangential forces F_n, F_s acting on each side of the interface must be equal to ensure internal equilibrium. Then we compute averaged values of the normal and tangential forces F_n^h , F_s^h of the finite element approximation, which for each side L_i of an element e_i can be computed by integrating the stresses σ_n^h, τ_{ns}^h over the sides, and noting that these stresses are constant along the sides, we obtain

$$F_n^h\Big|_{L_i}^{e_i} = \sigma_n^h\Big|_{L_i}^{e_i} \, t\, L \tag{2.53.a}$$

$$F_s^h\Big|_{L_i}^{e_i} = \tau_{ns}^h\Big|_{L_i}^{e_i} \, t\, L \tag{2.53.b}$$

where t is the thickness of the element and L is the length of the side.

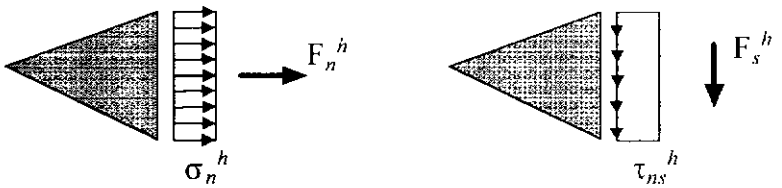


Figure 2.11. Resultant forces of the original stress field.

Then for each interior side L_i we compute an averaged value of the normal and tangential forces as

$$F_n^*\Big|_{L_i} = \frac{1}{2}\left(F_n^h\Big|_{L_i}^{e_1} + F_n^h\Big|_{L_i}^{e_2}\right) \tag{2.54.a}$$

$$F_s^*\Big|_{L_i} = \frac{1}{2}\left(F_s^h\Big|_{L_i}^{e_1} + F_s^h\Big|_{L_i}^{e_2}\right) \tag{2.54.b}$$

where e_1, e_2 are adjacent elements.

If we have prescribed normal tractions $\sigma_n = \overline{\sigma}_n$ or prescribed tangential tractions $\tau_{ns} = \overline{\tau}_{ns}$ at a boundary side we obtain the acting forces by integrating the tractions and impose the values to the respective averaged forces, that is

$$F_n^*\Big|_{L_i} = \int_{L_i} \overline{\sigma}_n \, t \, ds \tag{2.55.a}$$

$$F_s^*\Big|_{L_i} = \int_{L_i} \overline{\tau}_{ns} \, t \, ds \tag{2.55.b}$$

Assuming that the recovered normal and tangential tractions σ_n^*, τ_{ns}^* varies linearly on each side L_i (Fig. 2.11), we can compute the averaged values of these tractions at mid-side points of the boundary sides as

$$\sigma_n^*\big|_{L_i} = \frac{F_n^*\big|_{L_i}}{t\,L} \tag{2.56.a}$$

$$\tau_{ns}^*\big|_{L_i} = \frac{F_s^*\big|_{L_i}}{t\,L} \tag{2.56.b}$$

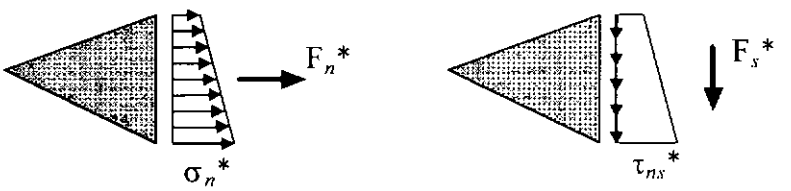


Figure 2.12. Resultant forces of the smoothed stress field.

Then we have for each triangle the averaged values σ_n^*, τ_{ns}^* of the normal and tangential tractions at mid-side points. To define completely the state of stress at these points we must compute an averaged value for the stress component σ_s^h at mid-side points. From the stress-strain relations (Eqn. 2.52.a), we have for each element e_i that the stress component σ_s^h on its side L_i is

$$\sigma_s^h\big|_{L_i}^{e_i} = \frac{E}{1-\nu^2}\,(\varepsilon_s^h\big|_{L_i} + \nu\,\varepsilon_n^h\big|_{L_i}) \tag{2.57}$$

where $\varepsilon_s^h\big|_{L_i}$, $\varepsilon_n^h\big|_{L_i}$ are the elongation strains at mid-side points of the finite element approximation. Then to obtain an averaged value of the stress σ_s^h we must compute averaged values of these elongations. In particular the tangential elongation $\varepsilon_s^h\big|_{L_i}$, from Eqn 2.51.a, can be written as

$$\varepsilon_s^h\big|_{L_i} = \left.\frac{\partial u_s^h}{\partial s}\right|_{L_i} = \frac{u_{s2}^h - u_{s1}^h}{L} \tag{2.58}$$

where u_s^h are the tangential displacements of side L_i obtained from the approximate solutions u_h, v_h, and u_{s1}^h, u_{s2}^h are the tangential displacements at the extremes of the side L_i whose length is L. It must be noted that these tangential strains are identical for adjacent elements and are superconvergent at mid-side points. Then there is no need to compute an averaged value of the tangential elongation between adjacent elements since it can be directly computed for each element e_i from its nodal displacements as

$$\varepsilon_s^*\big|_{L_i} = \left.\frac{\partial u_s^h}{\partial s}\right|_{L_i}^{e_i} = \left.\frac{u_{s2}^h - u_{s1}^h}{L}\right|_{L_i}^{e_i} \tag{2.59}$$

Also, note that for small deformations the rigth hand side of this equation represents the relative change of length of side L_i.

Then from the stress-strain relation for the normal stress σ_n (Eqn. 2.52.b), we have

$$\varepsilon_n^*\Big|_{L_i} = \frac{1-\nu^2}{E}\,\sigma_n^*\Big|_{L_i} - \nu\,\varepsilon_s^*\Big|_{L_i} \tag{2.60}$$

and making the substitution in Eqn.(2.57) we obtain the desired averaged value for the stress σ_s^* at the mid-side point as

$$\sigma_s^*\Big|_{L_i} = \nu\,\sigma_n^*\Big|_{L_i} + E\,\varepsilon_s^*\Big|_{L_i} \tag{2.61}$$

Note that there is no need to store this averaged value since it can be obtained from previously computed quantities. This implies that only two averaged values per side must be stored, one for the normal stresses and another for the tangential stresses. Note that if we were using *simple side averaging* we must store three averaged values per side.

With the averaged stresses in local axes we can compute the stresses in the direction of the global axes rotating the stress tensor as

$$\sigma_x^*\Big|_{L_i} = \sigma_s^*\Big|_{L_i}(\cos\alpha)^2 + \sigma_n^*\Big|_{L_i}(\sin\alpha)^2 - 2\,\tau_{ns}^*\Big|_{L_i}\sin\alpha\cos\alpha \tag{2.62.a}$$

$$\sigma_y^*\Big|_{L_i} = \sigma_s^*\Big|_{L_i}(\sin\alpha)^2 + \sigma_n^*\Big|_{L_i}(\cos\alpha)^2 + 2\,\tau_{ns}^*\Big|_{L_i}\sin\alpha\cos\alpha \tag{2.62.b}$$

$$\tau_{xy}^*\Big|_{L_i} = (\sigma_s^*\Big|_{L_i} - \sigma_n^*\Big|_{L_i})\sin\alpha\cos\alpha + \tau_{ns}^*\Big|_{L_i}[(\cos\alpha)^2 - (\sin\alpha)^2] \tag{2.62.c}$$

Finally we can interpolate these values of the stresses over each triangle with linear shape functions to define the smoothed fields as

$$\sigma_x^* = \sum_{i=1}^{3} M_i\,\sigma_x^*\Big|_{L_i} \tag{2.63.a}$$

$$\sigma_y^* = \sum_{i=1}^{3} M_i\,\sigma_y^*\Big|_{L_i} \tag{2.63.b}$$

$$\tau_{xy}^* = \sum_{i=1}^{3} M_i\,\tau_{xy}^*\Big|_{L_i} \tag{2.63.c}$$

where the functions M_i are the same shape functions employed in the scalar case (Eqn. 2.44), which are only continuos at mid-side points.

2.4.5.3 Flux Averaging on Transition Sides

For scalar problems we impose a balance of flux at the transition side (Fig. 2.12).

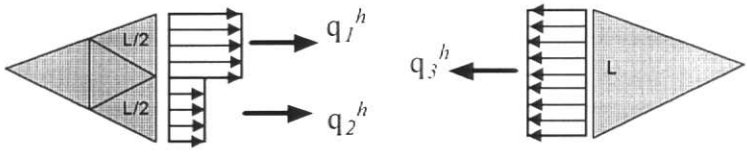


Figure 2.13. Flux balance at a transition interface.

We define the resultant fluxes from the finite element approximation over each side as

$$q_1^h = \left.\frac{\partial u_h}{\partial n}\right|_{L_1}^{e_1} L/2 \tag{2.64.a}$$

$$q_2^h = \left.\frac{\partial u_h}{\partial n}\right|_{L_1}^{e_2} L/2 \tag{2.64.b}$$

$$q_3^h = \left.\frac{\partial u_h}{\partial n}\right|_{L_1}^{e_3} L \tag{2.64.c}$$

where L is the length of the transition side and defining the unbalance flux Δq^h as

$$\Delta q^h = q_3^h - q_1^h - q_2^h \tag{2.65}$$

then we equilibrate the interface by equidistributing the unbalance flux Δq^h between both sides of the interface, resulting

$$q_1^* = q_1^h + \Delta q^h / 4 \tag{2.66.a}$$

$$q_2^* = q_2^h + \Delta q^h / 4 \tag{2.66.b}$$

$$q_3^* = q_3^h - \Delta q^h / 2 \tag{2.66.c}$$

Finally we can obtain averaged values of the normal derivatives at the mid-side of each element as

$$\left.\frac{\partial u^*}{\partial n}\right|_{L_1}^{e_1} = q_1^* \; 2/L \tag{2.67.a}$$

$$\left.\frac{\partial u^*}{\partial n}\right|_{L_i}^{e_2} = q_2^* \; 2 / L \tag{2.67.b}$$

$$\left.\frac{\partial u^*}{\partial n}\right|_{L_i}^{e_3} = q_3^* \; / \; L \tag{2.67.c}$$

For stress analysis we apply the same procedure to recover the averaged normal and tangential stresses substituting the resultant fluxes q_i by the resultant forces F_{ni}, F_{si} respectively.

2.4.5.4 Concentrated Loads

It must be noted that the presence of concentrated loads on nodes is not taken into account by the integration of boundary loads over the external sides (Eqns. 2.55), so no improvement of the smoothed gradient field near this loads is obtained. In general, concentrated loads are idealizations for high concentrated stresses on a small, but finite, region of the body. Then to improve the representation of the smoothed stress field around these singularities we substitute the concentrated loads by an equivalent triangular distribution of stresses over the sides surrounded the node (Fig. 2.14).

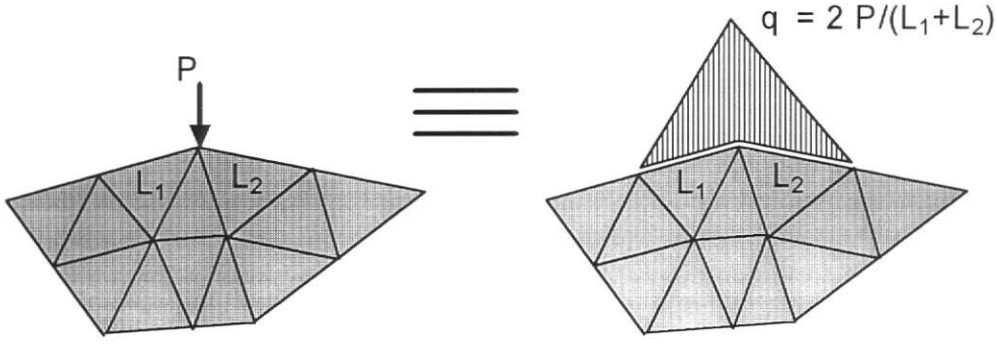


Figure 2.14. Equivalent stress distribution for concentrated loads.

When the refinement advance the length of the adjacement sides decreases and consequently the equivalent stress raises. In some physical situations the load can be considered as acting only on one side, and consequently we consider a triangular distribution on one side only.

2.4.5.5 Boundary Flux Computations

In portions of the boundary where the displacements are prescribed a poor representation of the stresses is obtained when they are computed from the derivatives of the finite element solution. Since in these portions of the boundary the smoothed stress field will adopt this values a poor estimation of the error is obtained. Then is convenient to use a better representation of the boundary flux for the error estimator. Here we will employ a

scheme of computing improved boundary stresses which is a simplified version of those proposed by Carey et al. [58] and Mizukami [59].

Basically the method consists in compute an equivalent stress distribution for the reaction forces concentrated at nodes. These forces are obtained from the finite element computations as the difference between internal and external forces in all the nodes of the discretization. Then these concentrated forces are treated in the same way as discussed in the previous section, substituting them by an equivalent triangular distribution of stresses. But in this case we make the distinction between normal and tangential forces, since boundary conditions on sides are specified in such directions. Then, we compute for each boundary node averaged values for the normal and tangential reactions at each adjacent side (Fig.2.15) as

$$Rn_m = (Rn_1 + Rn_2) / 2 \tag{2.68.a}$$

$$Rs_m = (Rs_1 + Rs_2) / 2 \tag{2.68.b}$$

where Rn_1, Rn_2 are the projections of the reaction force R in the direction of the normal of each side, and Rs_1, Rs_2 are the projections in the direction of the tangent of each side.

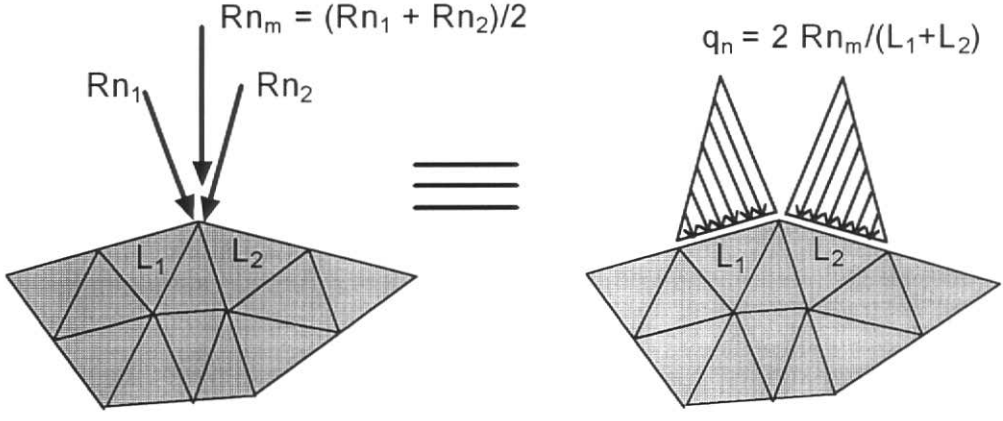


Figure 2.15. Equivalent stress distribution for normal reaction load.

In some physical situations the load must be considered as acting only on one side, and consequently we consider a triangular distribution on one side only. Such a case occurs, for example, when adjacent sides have different boundary conditions in the same direction.

2.4.6 Estimators for Non-Linear Elasticity

We restrict the applications to problems with large displacements and rotations but small elastic deformations. This type of problems are generally known as '*geometrical non-linear*' problems.

The classical definition of the tensor strain as a linear function of the derivatives of the solution (Eqns. 2.46) is limited to problems with very small displacements and deformations. For this type of linear problems the energy of deformation is given by

$$U = \tfrac{1}{2}\left(\int_{\Omega_0} (\varepsilon_L)^T \mathbf{D}\ (\varepsilon_L)\ d\Omega_0\right) \tag{2.69}$$

where **D** is the constituve matrix of elastic coefficients, Ω_0 is the volume of the initial undeformed configuration, and ε_L is the vector of linear strains

$$(\varepsilon_L)^T = \{\ \varepsilon_x \quad \varepsilon_x \quad \gamma_{xy}\ \} \tag{2.70}$$

For problems with large displacements more general definitions of strains are needed. A very general definition of strain is given by the Green's strain tensor [52] whose componentes for plane problems are

$$E_x = \frac{\partial u}{\partial x} + \frac{1}{2}\left[\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2\right] \tag{2.71.a}$$

$$E_y = \frac{\partial v}{\partial y} + \frac{1}{2}\left[\left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2\right] \tag{2.71.b}$$

$$E_{xy} = \frac{1}{2}\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) + \frac{1}{2}\left[\frac{\partial u}{\partial x}\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\frac{\partial v}{\partial y}\right] \tag{2.71.c}$$

where the derivatives and displacements $u(x_0,y_0)$ are referred to the initial undefomed configuration. It can be proved that if actual deformations are small this strains have the meaning of infinitesimal strains referred to an undeformed configuration that rotates and translates with the body [53,11]. Then, assuming elastic behaviour, the energy of deformation of the current deformed configuration can be expressed as

$$U = \tfrac{1}{2}\left(\int_{V_0} (\varepsilon_{NL})^T \mathbf{D}\ (\varepsilon_{NL})\ dV_0\right) \tag{2.72}$$

where ε_{NL} is the vector of non-linear strains defined as

$$(\varepsilon_{NL})^T = \{\ E_x \quad E_x \quad 2\,E_{xy}\ \} \tag{2.73}$$

Also, for very small elastic deformations the stresses referred to the undeformed configuration can be approximated as [11,52]

$$\tilde{\sigma} \approx \boldsymbol{D}\,\varepsilon_{NL} \tag{2.74}$$

In an adaptive analysis we are interested in an accurate description of the state of stress of the current deformed configuration. Then for compute estimates of an error function in the current deformed configuration we use the next energy norm for each element *i* :

$$\|e^*\|_i = \left(\int_{\Omega_i} (\tilde{\sigma}^* - \tilde{\sigma}_h)^T \, \mathbf{D}^{-1} \, (\tilde{\sigma}^* - \tilde{\sigma}_h) \, d\Omega_0 \right)^{1/2} \tag{2.75}$$

Then for the computation of the *flux-projection* based error estimators for geometrically nonlinear analysis we use the stresses given by Eqn. (2.74) and the energy norm computed as in Eqn. (2.75).

For the particular case of the side averaging error estimator we need to compute the relative elongation $\varepsilon_s^*|_{L_i}$, which for small strains can be estimated as

$$\varepsilon_s^*|_{L_i} = \frac{L_i^1 - L_i^0}{L_i^0} \tag{2.76}$$

where L_i^1 is the current length of side L_i computed in the deformed configuration and L_i^0 is the initial undeformed length of the same side.

2.5. Numerical Comparisons of Error Estimators

We present numerical examples of plane elasticity in order to compare the reliability of the error estimators when used as the basis of an adaptive process.

We compare two disposition of sampling points for linear triangles in the superconvergent patch recovery technique: the standard one with sampling points at centroids and other with sampling points located at midsides (Fig. 2.16).

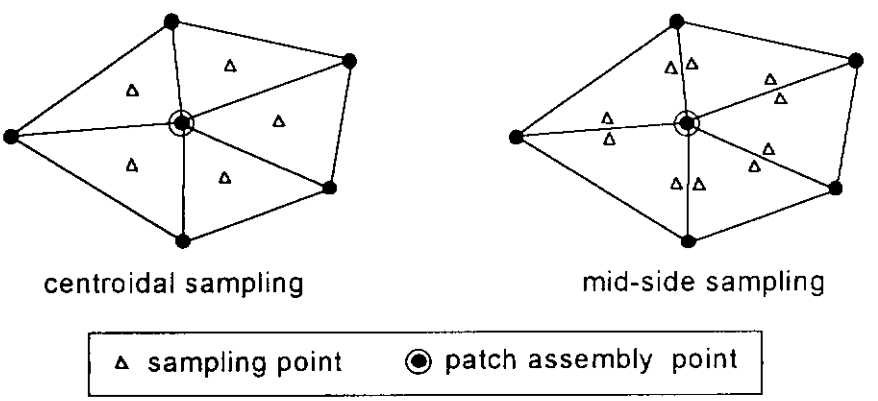


Figure 2.16 Different samplings for the superconvergent patch recovery technique on linear triangles.

In all the examples one protective layer of refined elements was added sorrounding the elements that need to be refined.

2.5.1 Short Cantilever Beam

We considered a short cantilever beam under plane strain conditions. Poisson's ratio and Young's modulus were taken to be 0.3 and 1.0 respectively. The geometry, loads, boundary conditions and the initial mesh are shown in Fig. 2.17. We have taken as reference solution that given by Ainsworth et al.[56] where the square energy norm of the exact solution is given as $\|\boldsymbol{u}\|^2 = 1.903697$.

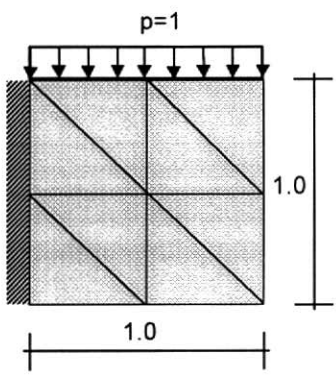


Figure 2.17. Short cantilever beam under plane strain conditions.

The performance of the error estimators is shown in Tables 2.1, to 2.5, where the columns are defined as follows:

DOF: degrees of freedom of the mesh.

$\|\boldsymbol{u}_h\|^2$: square of the energy norm of the finite element solution.

$\|\boldsymbol{e}^*\|^2$: square of the energy norm of the estimated error.

$\|\boldsymbol{e}_{ex}\|^2$: square of the energy norm of the exact error.

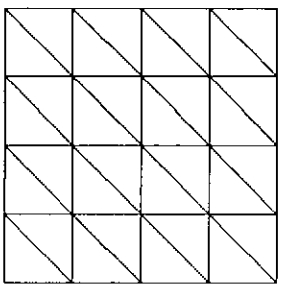
θ : effectivity index of the error estimator.

η^* % : estimated relative error percent.

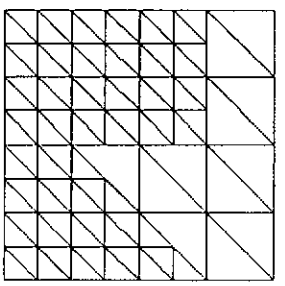
η_{ex} % : exact relative error percent.

In Figure 2.18 we can see the sequence of adaptive refinements to achieve 5 per cent accuracy in the energy norm using side flux averaging as error estimator and in Fig. 2.19 the sequence obtained using the superconvergent patch recovery technique. The exact relative errors are also indicated for each mesh. We can observe that both estimators guides the refinement concentrating more elements at corners where high gradients of stressses occurs. But the superconvergent patch recovery concentrates more elements near the boundaries, overestimating the error locally in this regions.

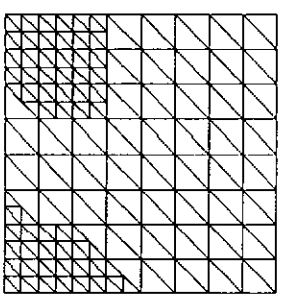
For this example, the superconvergent patch recovery technique with sampling points at mid-sides gives the same sequence of meshes that when using centroidal sampling. This confirm the fact that this technique is rather insensitive to the selection of sampling points, as noted by Babuska et al.[34]. Also, the nodal averaging estimator gives the same meshes that those obtained with the superconvergent patch recovery technique, this can be due to the regularity of the meshes. The differences between these estimators can be observed in the column of the square norm of the estimated error $\|e^*\|^2$ in Tables 2.2 to 2.4, where we can see that the differences are minimal.



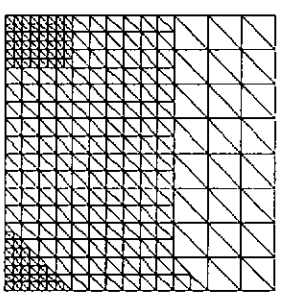
MESH 1 η = 35.70 %
DOF = 50



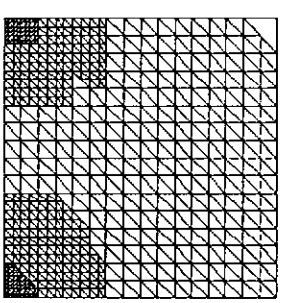
MESH 2 η = 24.52 %
DOF = 126



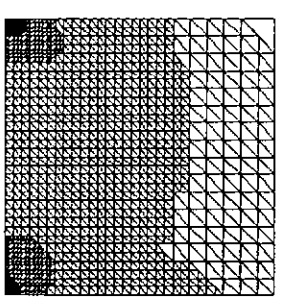
MESH 3 η = 16.48 %
DOF = 274



MESH 4 η = 11.18 %
DOF = 578



MESH 5 η = 7.71 %
DOF = 1134



MESH 6 η = 5.19 %
DOF = 2358

Figure 2.18. Short cantilever beam. Side flux averaging error estimator.

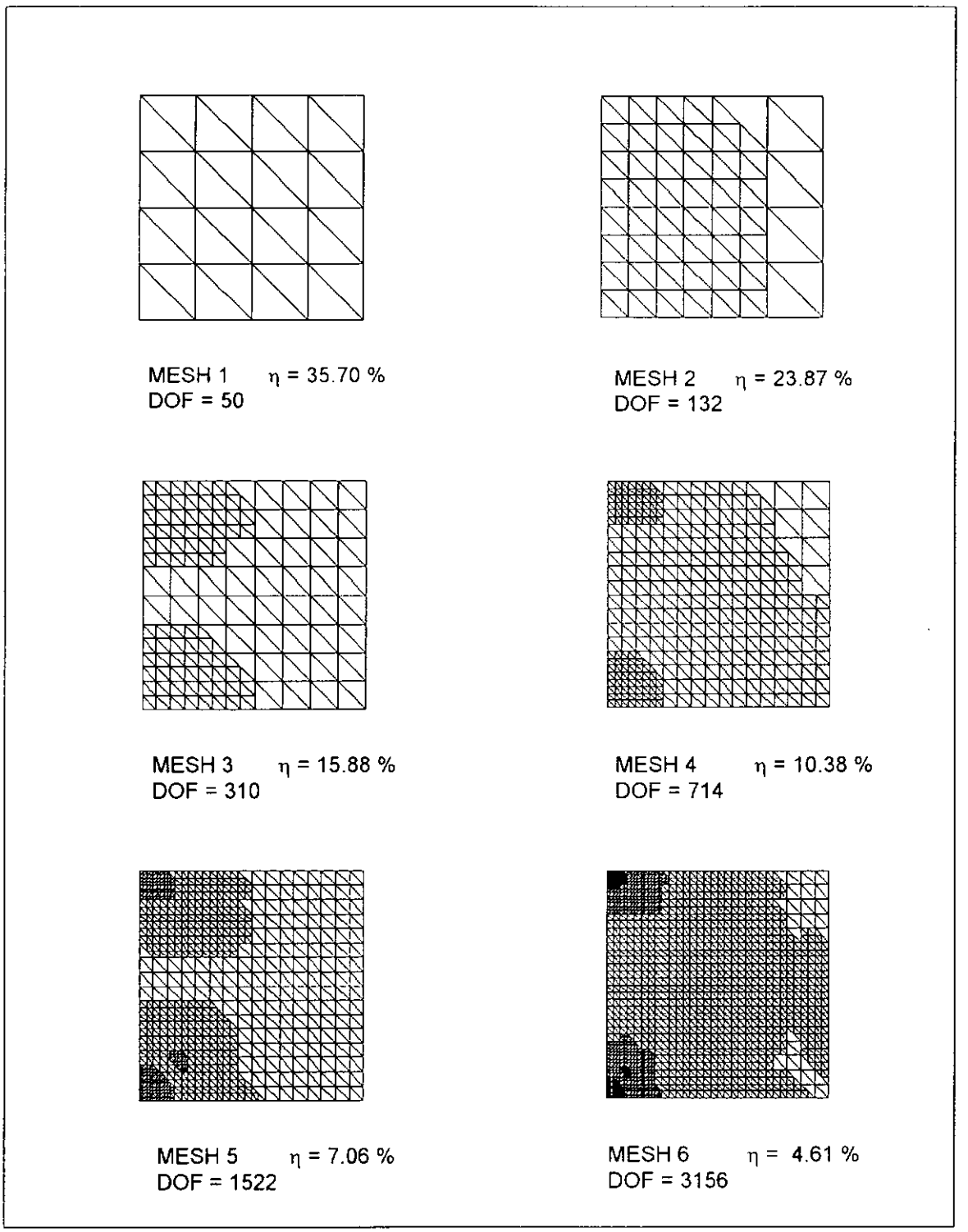


Figure 2.19. Short cantilever beam. Superconvergent Patch Recovery error estimator. Centroidal sampling.

Table 2.1. Short cantilever beam under plane strain conditions. Error estimator: Side flux averaging .

mesh	DOF	$\|u_h\|^2$	$\|e^*\|^2$	$\|e_{ex}\|^2$	θ	η* %	η_{ex} %
0	18	1.41467	0.329554	0.489027	0.82	43.47	50.68
1	50	1.66112	0.195492	0.242574	0.90	32.45	35.70
2	126	1.78924	0.963643 (-1)	0.114453	0.92	22.61	24.52
3	274	1.85202	0.455103 (-1)	0.516793 (-1)	0.94	15.49	16.48
4	578	1.87991	0.209014 (-1)	0.237916 (-1)	0.94	10.49	11.18
5	1134	1.89239	0.100609 (-1)	0.113041 (-1)	0.94	7.27	7.71
6	2358	1.89856	0.461769 (-2)	0.513505 (-2)	0.95	4.93	5.19

Table 2.2. Short cantilever beam under plane strain conditions. Error estimator: Superconvergent Patch Recovery. Centroidal sampling.

mesh	DOF	$\|u_h\|^2$	$\|e^*\|^2$	$\|e_{ex}\|^2$	θ	η* %	η_{ex} %
0	18	1.41467	0.195423	0.489027	0.63	34.84	50.68
1	50	1.66112	0.129638	0.242574	0.73	26.91	35.70
2	132	1.79522	0.701769 (-1)	0.108478	0.80	19.40	23.87
3	310	1.85570	0.351254 (-1)	0.479999 (-1)	0.86	13.63	15.88
4	714	1.88320	0.153225 (-1)	0.204928 (-1)	0.86	8.98	10.38
5	1522	1.89422	0.754693 (-2)	0.947995 (-2)	0.89	6.30	7.06
6	3156	1.89965	0.337587 (-2)	0.404689 (-2)	0.91	4.21	4.61

Table 2.3 Short cantilever beam under plane strain conditions. Error estimator: Superconvergent Patch Recovery. Mid-side sampling.

mesh	DOF	$\|u_h\|^2$	$\|e^*\|^2$	$\|e_{ex}\|^2$	θ	η* %	η_{ex} %
0	18	1.41467	0.195678	0.489027	0.63	34.86	50.68
1	50	1.66112	0.129947	0.242574	0.73	26.94	35.70
2	132	1.79522	0.699190 (-1)	0.108478	0.80	19.36	23.87
3	310	1.85570	0.351438 (-1)	0.479999 (-1)	0.86	13.63	15.88
4	714	1.88320	0.153169 (-1)	0.204928 (-1)	0.86	8.98	10.38
5	1522	1.89422	0.754636 (-2)	0.947995 (-2)	0.89	6.30	7.06
6	3156	1.89965	0.337434 (-2)	0.404689 (-2)	0.91	4.21	4.61

Table 2.4. Short cantilever beam under plane strain conditions. Error estimator: Nodal averaging.

mesh	DOF	$\|u_h\|^2$	$\|e^*\|^2$	$\|e_{ex}\|^2$	θ	η* %	η_{ex} %
0	18	1.41467	0.195423	0.489027	0.63	34.84	50.68
1	50	1.66112	0.129638	0.242574	0.73	26.91	35.70
2	132	1.79522	0.693323 (-1)	0.108478	0.80	19.28	23.87
3	310	1.85570	0.350603 (-1)	0.479999 (-1)	0.86	13.62	15.88
4	714	1.88320	0.152918 (-1)	0.204928 (-1)	0.86	8.97	10.38
5	1522	1.89422	0.754271 (-2)	0.947995 (-2)	0.89	6.30	7.06
6	3156	1.89965	0.337102 (-2)	0.404689 (-2)	0.91	4.21	4.61

Table 2.5 Short cantilever beam under plane strain conditions. Error estimator: Simple side averaging (without external fluxes).

mesh	DOF	$\|u_h\|^2$	$\|e^*\|^2$	$\|e_{ex}\|^2$	θ	η* %	η_{ex} %
0	18	1.41467	0.120414	0.489027	0.50	28.01	50.68
1	50	1.66112	0.923266 (-1)	0.242574	0.62	22.95	35.70
2	126	1.79093	0.501198 (-1)	0.112766	0.67	16.50	24.34
3	298	1.84919	0.271306 (-1)	0.545053 (-1)	0.71	12.02	16.92
4	676	1.87784	0.123036 (-1)	0.258560 (-1)	0.69	8.07	11.65
5	1412	1.88888	0.623049 (-2)	0.148162 (-1)	0.65	5.73	8.82
6	3098	1.89422	0.269558 (-2)	0.947500 (-2)	0.53	3.77	7.05

Comparing Tables 2.1 and 2.2 we can observe that the side flux averaging estimator gives effectivity indices more near the unity that the superconvergence patch recovery technique, this is displayed in Fig. 2.20 where it is also shown the history of the effectivity index of the simple side averaging estimator without taken into account the external fluxes. It can be seen that in this last case the estimator looses its asymptotic exactness and diverges with increasing refinement. The same phenomena has been noted in some meshes by Babuska and Rodriguez [61] for an error estimator developed by Zienkiewicz and Zhu [29] based on stress projections. This suggests that the consideration of external fluxes in the development of error estimators by projection migth be an essential factor to obtain reliable results.

It must be noted that all the estimators underestimate the global error, as indicated by effectivity indices with values less than unity, but the side averaging estimator predicts a better distribution of local errors which results in a reduction of the number of elements needed to reach the desired accuracy, as can be seen in the Tables.

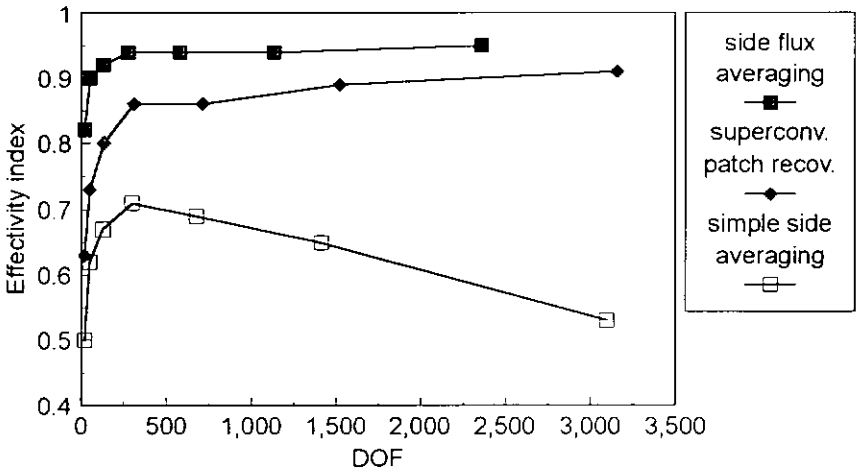


Figure 2.20. Short cantilever beam. Comparison of effectivity indices.

We can see in Fig. 2.21 that an optimal convergence rate, which is 0.5 in this case, is achieved by the adaptive process at intermediate and final stages of the refinement, using either side flux averaging or the superconvergent patch recovery technique as error estimators.

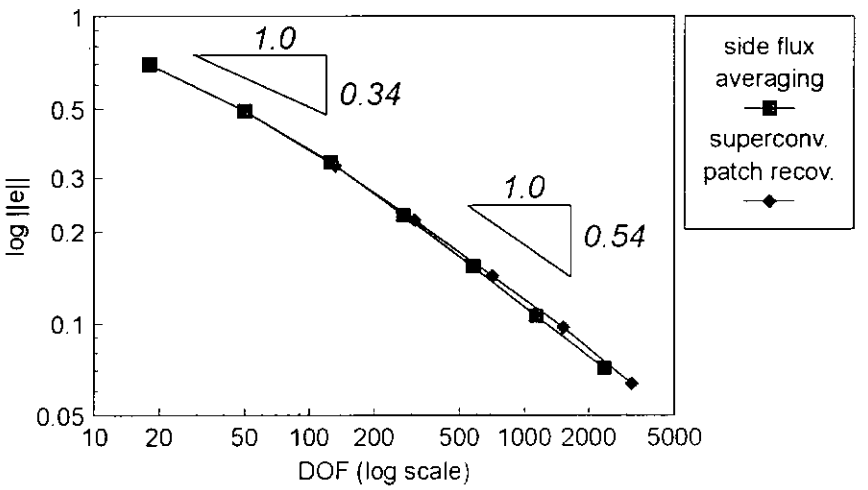


Figure 2.21. Short cantilever beam. Convergence rates.

Finally, we can see in Figs. 2.22 and 2.23 the band plots for the recovered stress σ_x^* for meshes 1 and 6, obtained by smoothing with side flux averaging. It must be noted that this stress field is only continuous at mid-sides of elements and severe discontinuities can be observed at early stages of the refinement process as shown in Fig. 2.22, but these discontinuities are imperceptible in the final mesh, as it can be seen in Fig. 2.23. Note that these plots have several practical implications because they gives a qualitative indication of the error which complements the more abstract measure of the error in the energy norm.

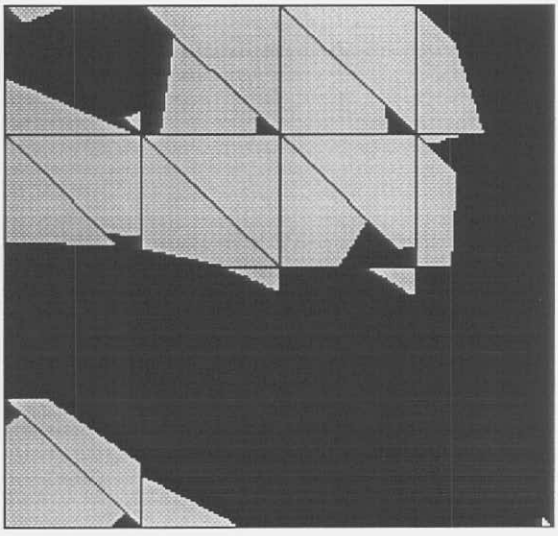


Figure 2.22. Short cantilever beam. Band plot of smoothed stress σ_x* on mesh 1. (Predicted error in energy norm η* = 22.64%)

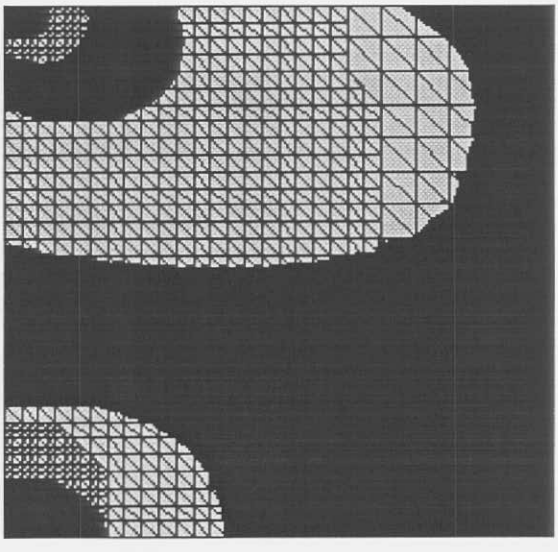


Figure 2.23. Short cantilever beam. Band plot of smoothed stress σ_x* on mesh 6. (Predicted error in energy norm η* = 4.93%)

2.5.2 L-Shaped Region in Plane Stress

We considered an standard L-shaped domain problem under plane stress conditions. Poisson's ratio and Young's modulus were taken to be 0.3 and 100000.0 respectively. The geometry, loads, boundary conditions and the initial mesh are shown in Fig. 2.24. We have taken as reference solution that given by Fish et al. [60] where the square energy norm of the exact solution is taken $\|u\|^2 = 0.311333333$.

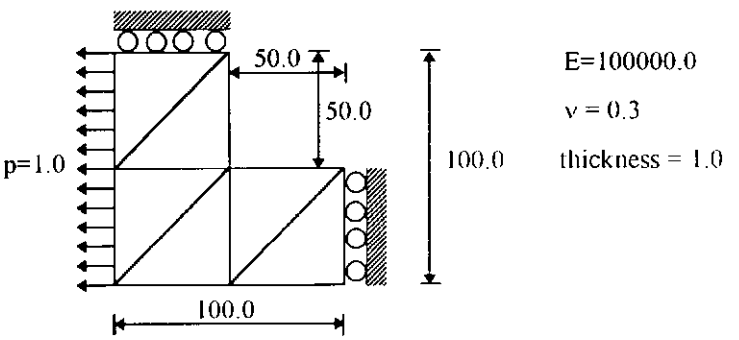
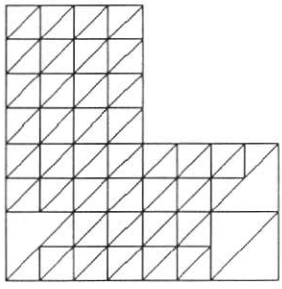


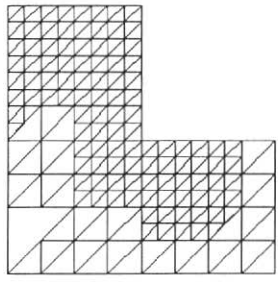
Figure 2.24 - L-shape domain problem.

The final sequence of meshes obtained by adaptive refinement to achieve 5 per cent accuracy in the energy norm can be seen in Figure 2.25 for the side flux averaging method and in Figure 2.26 for the superconvergent patch recovery technique. The exact relative errors are also indicated for each mesh. We can observe that both estimators guides the refinement concentrating more elements at corners where high gradients of stressses occurs. But, as in the previous example, the superconvergent patch recovery error estimator concentrates more elements near the boundaries, overestimating the error locally in this regions.

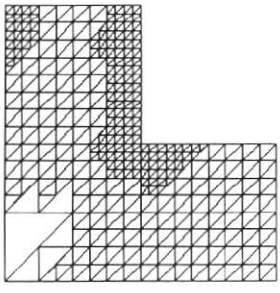
The performance of the error estimators is shown in Tables 2.6 to 2.8, where we can observe that the side flux averaging estimator gives effectivity indices more near the unity that the superconvergence patch recovery technique, this is displayed in Fig. 2.27 where the history of the effectivity index of the simple side averaging estimator without taken into account the external fluxes it is shown too. This estimator performs well in this case, but as it was seen in the previous example of the short cantilever beam it is not very reliable. Therefore the side flux averaging method with consideration of external fluxes seems to be the better choice.



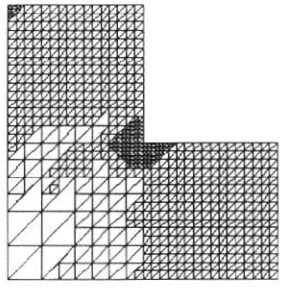
MESH 2 η = 34.46 %
DOF = 118



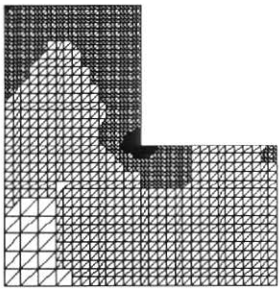
MESH 3 η = 21.99 %
DOF = 318



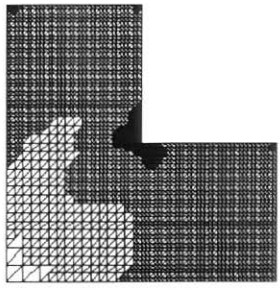
MESH 4 η = 15.53 %
DOF = 686



MESH 5 η = 10.46 %
DOF = 1502



MESH 6 η = 7.18 %
DOF = 3086



MESH 7 η = 4.81 %
DOF = 6222

Figure 2.25. L-shaped domain. Side flux averaging error estimator.

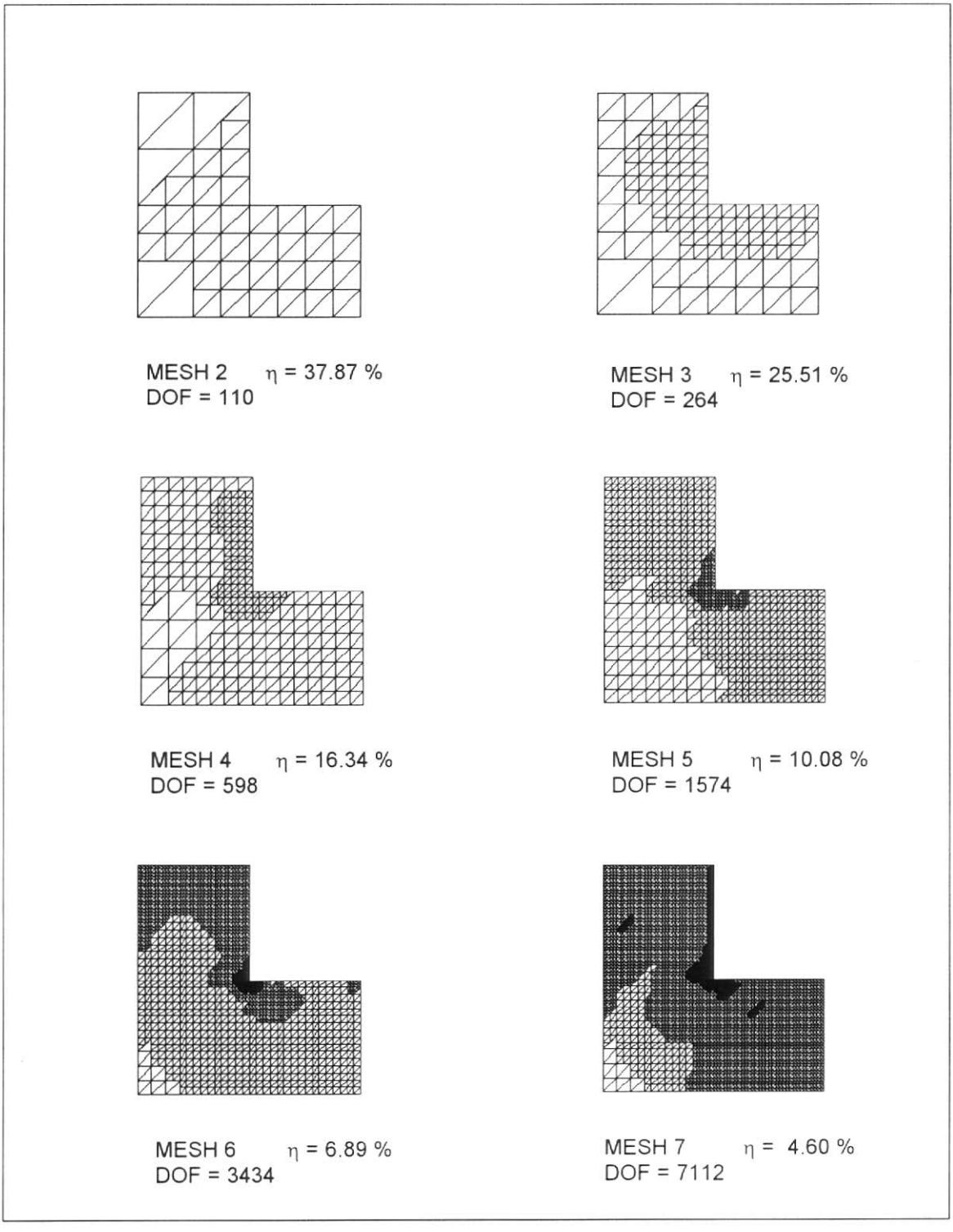


Figure 2.26. L-shaped domain. Superconvergent Patch Recovery error estimator. Centroidal sampling.

Table 2.6. L-shaped domain. Error estimator: Side flux averaging .

mesh	DOF	$\|u_h\|^2$	$\|e^*\|^2$	$\|e_{ex}\|^2$	θ	η* %	η_{ex} %
0	16	0.197501	0.347333 (-1)	0.113832	0.55	38.67	60.47
1	38	0.229759	0.324351 (-1)	0.815741 (-1)	0.63	35.17	51.19
2	118	0.274371	0.240250 (-1)	0.369623 (-1)	0.81	28.37	34.46
3	318	0.296282	0.121311 (-1)	0.150514 (-1)	0.90	19.83	21.99
4	686	0.303826	0.641430 (-2)	0.750773 (-2)	0.92	14.38	15.53
5	1502	0.307928	0.299014 (-2)	0.340551 (-2)	0.94	9.81	10.46
6	3086	0.309726	0.152136 (-2)	0.160698 (-2)	0.97	6.99	7.18
7	6222	0.310613	0.721558 (-3)	0.720684 (-3)	1.00	4.81	4.81

Table 2.7. L-shaped domain. Error estimator: Superconvergent Patch Recovery. Centroidal sampling.

mesh	DOF	$\|u_h\|^2$	$\|e^*\|^2$	$\|e_{ex}\|^2$	θ	η* %	η_{ex} %
0	16	0.197501	0.255084 (-1)	0.113832	0.47	33.82	60.47
1	38	0.223980	0.237872 (-1)	0.873536 (-1)	0.52	30.98	52.97
2	110	0.266689	0.242154 (-1)	0.446439 (-1)	0.74	28.85	37.87
3	264	0.291071	0.149150 (-1)	0.202619 (-1)	0.86	22.08	25.51
4	598	0.303017	0.697530 (-2)	0.831645 (-2)	0.92	15.00	16.34
5	1574	0.308168	0.271271 (-2)	0.316553 (-2)	0.93	9.34	10.08
6	3434	0.309853	0.135092 (-2)	0.147989 (-2)	0.96	6.59	6.89
7	7112	0.310675	0.630926 (-3)	0.658614 (-3)	0.98	4.50	4.60

Table 2.8 L-shaped domain. Error estimator: Simple side averaging (without external fluxes).

mesh	DOF	$\|u_h\|^2$	$\|e^*\|^2$	$\|e_{ex}\|^2$	θ	η* %	η_{ex} %
0	16	0.197501	0.162092 (-1)	0.113832	0.38	27.54	60.47
1	42	0.243336	0.238619 (-1)	0.679974 (-1)	0.59	29.88	46.73
2	130	0.280967	0.170255 (-1)	0.303663 (-1)	0.75	23.90	31.23
3	384	0.299512	0.754458 (-2)	0.118213 (-1)	0.80	15.68	19.49
4	934	0.305376	0.402265 (-2)	0.595693 (-2)	0.82	11.40	13.83
5	1850	0.308534	0.207502 (-2)	0.279883 (-2)	0.86	8.17	9.48
6	3974	0.310044	0.104259 (-2)	0.128922 (-2)	0.90	5.79	6.44
7	7730	0.310711	0.531589 (-3)	0.622042 (-3)	0.92	4.13	4.47

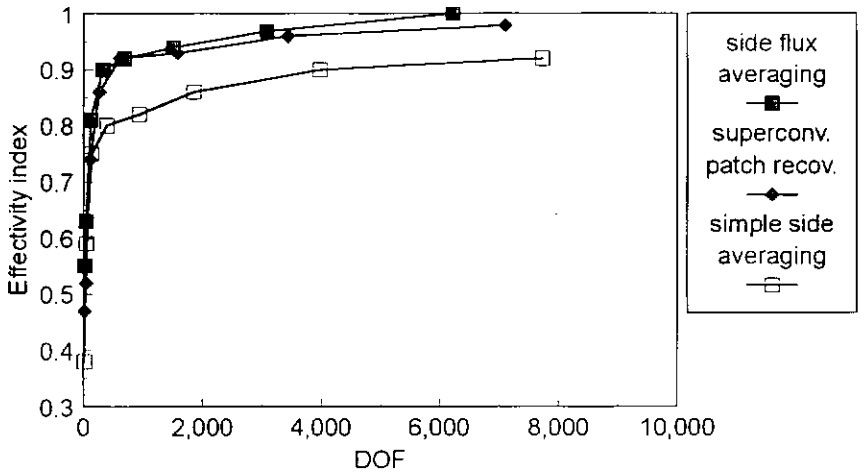


Figure 2.27. L-shaped domain. Comparison of effectivity indices.

We can see in Fig. 2.28 that an optimal convergence rate of 0.5 is achieved by the adaptive process at intermediate and final stages of the refinement, using either side flux averaging or the superconvergent patch recovery technique as error estimators.

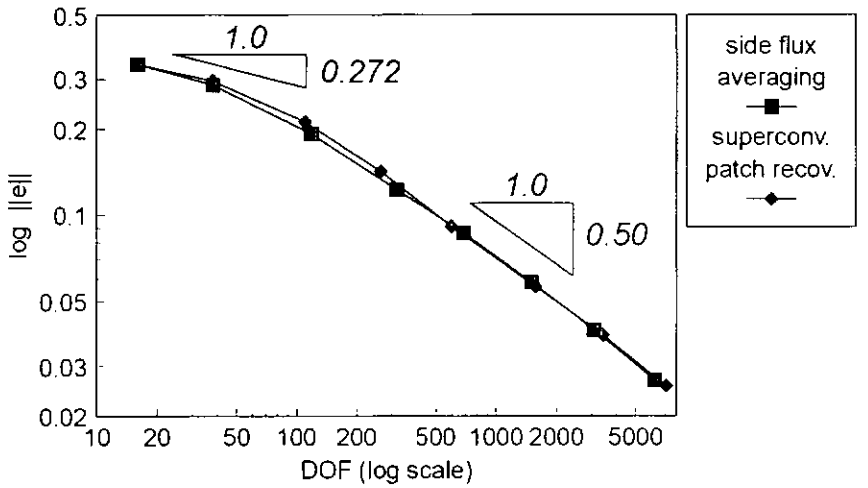


Figure 2.28. L-shaped domain. Convergence rates.

Finally, we can see in Figs. 2.29 and 2.30 the band plots for the recovered stress σ_x^* for meshes 2 and 7, obtained by smoothing with side flux averaging. As in the previous example we can observe severe discontinuities at early stages of the refinement process as shown in Fig. 2.29, but these discontinuities are imperceptible in the final mesh, as it can be seen in Fig. 2.30.

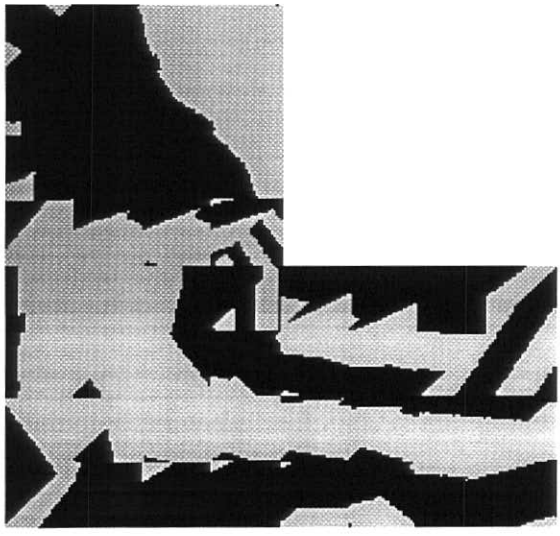


Figure 2.29. L-shaped domain. Band plot of smoothed stress σ_x* on mesh 2. (Predicted error in energy norm η* = 28.37%)

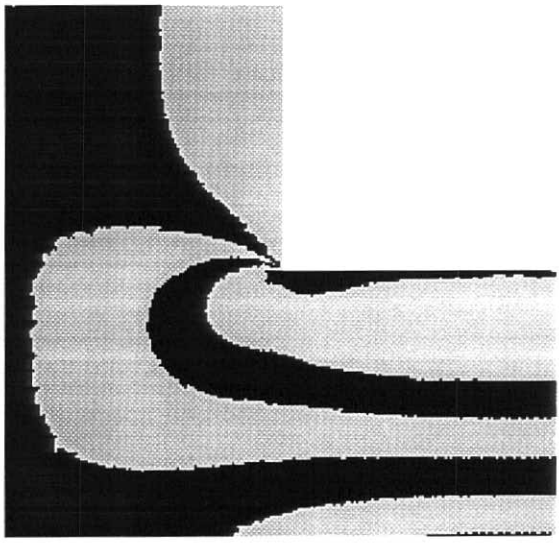


Figure 2.30. L-shaped domain. Band plot of smoothed stress σ_x* on mesh 7. (Predicted error in energy norm η* = 4.81%)

Chapter 3

Data Structures

3.1 Introduction

An important ingredient of a general purpose *h* adaptative program is the data structure. In designing a data structure, compromises must be made to conserve storage and avoid excessive computation time. In the *h*-refinement strategy here adopted an element is subdivided into four elements. These relationships among elements can be best expressed through a hierarchical or tree-type data structure.

The usual data structures used in non-adaptive finite element codes [11,12,13] are not directly applicable in adaptive codes, since there is a need of additional information about the refinement process. There are several data structures for adaptive finite element codes [5,6,7]. Some of them are based in a general labeled tree [5,6] which emphasizes a minimization of storage, but the traditional finite element data structure has been abandoned. In others [7], an augmented data structure is employed, consisting of traditional finite element arrays supplemented by a neighboring element connectivity, this type of data structure can be readily implemented into existing finite element codes. Here we adopt a data structure of this kind where the traditional nodal coordinate and element connectivity arrays are supplemented with additional arrays of connectivity information. In the present implementation we consider only refinements of the mesh, but the data structure was designed to permit the coarsening of group of elements with minimal modifications.

3.2 Basic Identification Arrays

Two-dimensional meshes consist of nodes, sides and elements. Beginning with an initial mesh or level zero the successive generated nodes, sides and elements are grouped into levels of refinement. Each node, side or element is identified with the level where it appears by the first time. Node, sides and elements are numbered individually in their levels.

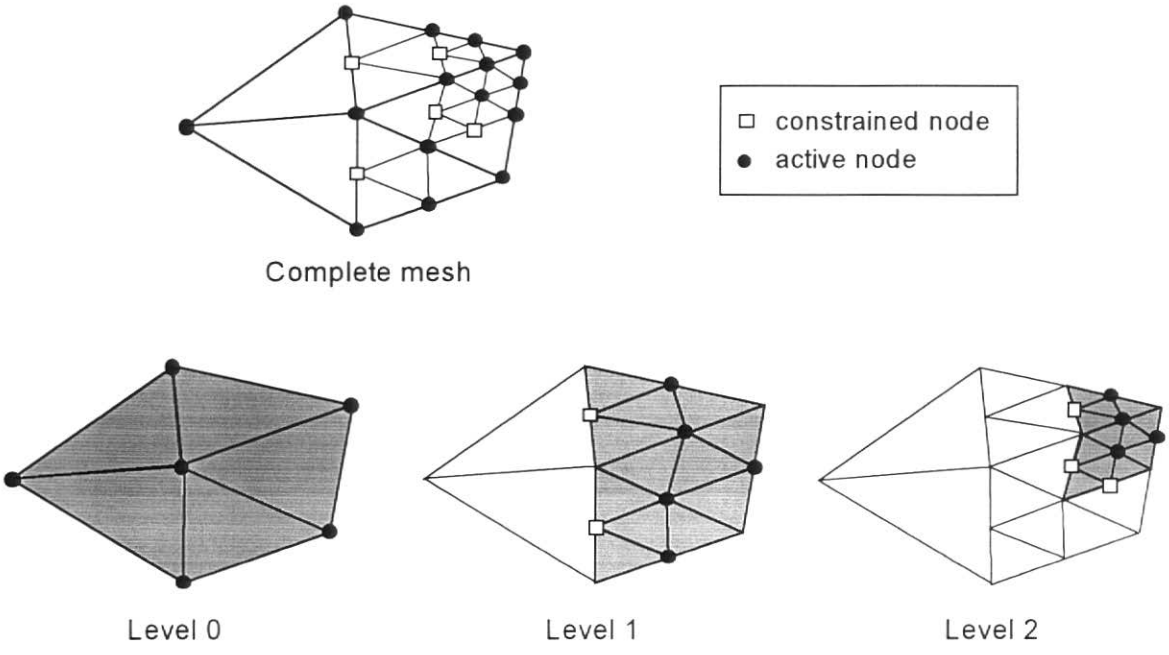


Figure 3.1. Level partition of the mesh.

The number of elements, sides and nodes of each level *curlev* are stored in the next variables:

numel[curlev] : *number of elements in level* ***curlev.***
sides[curlev] : *number of sides in level* ***curlev.***
nodes[curlev] : *number of nodes in level* ***curlev.***

To identify the incidences of sides and elements we define in each level *curlev* the next integer vectors:

side_nodes[curlev][i][nsn] : *nodal assignments of side i of level* ***curlev.***
elm_sides[curlev][j][nes] : *side assignments of element j of level* ***curlev.***

where nsn is the number of nodes per side and nes is the number of sides per element.

These are the basic arrays that define the topology of the mesh. In the vector **side_nodes**[curlev][i][nsn] we store the node numbers of each side, and in the vector **elm_sides**[curlev][j][nes] the side numbers counterclockwise, where a negative side number indicates that the node numbers of the respective side are given in opposite direction.

Since nodes are numbered individually in each level, it is necessary an additional vector to identify the level of the nodes of each side. In the vector **side_vlev**[curlev][i][nsn] we store the node levels of each side. Also, to facilite the operations over elements we define the vectors **elm_nodes**[curlev][j][nen] and **elm_vlev**[curlev][j][nen] which stores the node numbers and node levels of each element,

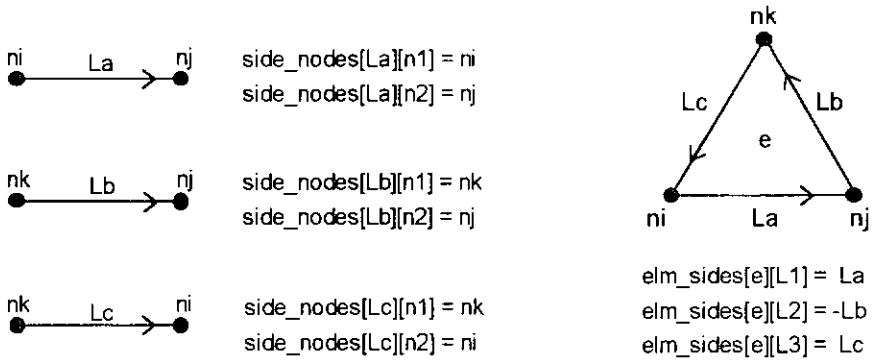


Figure 3.2. Incidences of sides and elements.

During each refinement of the mesh new elements and sides are generated, this implies in modifications in the data structures of each level. In the next sections these modifications are described. We define the '*current*' level as the level whose elements are refined, and we define the '*next*' level as the immediate superior level to the current level. Then all elements and sides generated by subdivision of elements and sides of the '*current*' level are located in the '*next*' level.

3.3 Classification of Elements of Current Level

For each level we have a list of elements of that level. Since we are not considered derefinements of the mesh, the size of this list can only growth or remain unchanged after each refinement. New elements are added to end of this list as the same time as they are generated. Elements are numbered independently in each level in correspondence with their position in the list of each level, we refer to this enumeration as the *'original'* enumeration of elements.

The elements of each level can be grouped into refinable and unrefinable elements. An element is considered unrefinable if it has an adjacent element of an inferior level, therefore, only are considered refinable those elements whose sides are adjacent to refined or unrefined elements of the same level. The number of refinable and unrefinable elements of level ***curlev*** are stored in the next variables:

Rnumel[curlev] : *refinable elements of level* ***curlev***.
Unumel[curlev] : *unrefinable elements of level* ***curlev***.

Between the group of refinable elements of each level we can distinguish refined and unrefined elements. The number of refined and unrefined elements of the group of refinable elements of level ***curlev*** are stored in the next variables:

FRnumel[curlev] : *refined elements of level* ***curlev.***
URnumel[curlev] : *refinable elements of level* ***curlev*** *not refined.*

In order to facilitate the generation of the mesh arrays of the next level we rearrange the elements of each level grouping first the refined elements, then the refinable elements not refined and then the unrefinable elements. The new ordering of elements of each level, is stored in the integer matrix **velm**[maxlev][numel]. We refer to this new identification as the *'modified'* enumeration. Then **velm**[curlev][ie] stores the original identification ***oe*** of the element of the level ***curlev*** identified as ***ie*** in the modified enumeration. This new enumeration is schematically represented in figure 3.3, where ascending element numbering is considered from left to right.

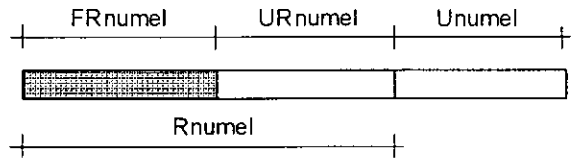


Figure 3.3. Element ordering of each level in array VELM [*curlev*].

In the initial level all the elements are considered refinable, so before the first refinement we have:

Rnumel[0] = **numel**[0]
FRnumel[0] = 0

URnumel[0] = **numel**[0]
Unumel[0] = 0

3.4 Modification of Element Ordering After Refinement of Current Level

After having determined the values of the error indicators in the refinable elements of level ***curlev***, all elements lying above a preset threshold are refined. To identify which refinable elements will be refined and which will remain unchanged we define an integer array **relm** for each level in analysis. For each refinable element ***oe*** not subdivided of the level ***curlev*** this vector can assume one of the next values:

relm[curlev][oe] = 0 for element oe not to be refined
relm[curlev][oe] = 1 for element oe to be refined

All the new refined elements pertains to the group of refinable elements. To store the number of refinable elements that are refined in each level and the number of refinable elements that remain unrefined in each level ***curlev*** we define the next variables:

CFRnumel[curlev] : *refinable elements that are refined in level* ***curlev***.
RURnumel[curlev] : *refinable elements that remain unrefined in level* ***curlev***.

In order to facilitate the creation of the mesh arrays of the next level we rearrange the elements of the recently refined level as indicated in Figure 3.4. We number first the existing refined elements *(FRnumel)*, then the refinable elements that are refined *(CFRnumel)* , then the refinable elments that remain unchanged *(RURnumel)*, and finally we left unchanged the unrefinable elements of the level *(Unumel)*.

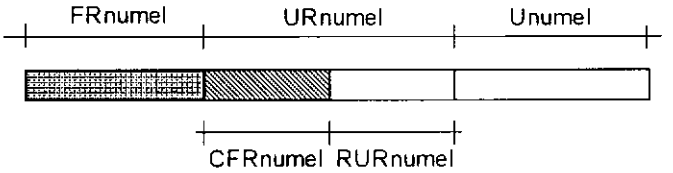


Figure 3.4. Element ordering after refinement in array VELM [*curlev*].

We must modify the integer matrix **velm**[curlev][numel] to store the new ordering of elements. To obtain the new ordering for the current level ***(curlev)*** we must analize the refinable elements of each level, to do this we employ the next algorithm written in C language:

```
i1 = FRnumel[curlev];

for (ie=i1+1 ; ie<= Rnumel[curlev] ; ie++)
  {
    if ( relm[curlev][velm[curlev][ie]] )
      {
        ++i1;
        e1 = velm[curlev][ie];
        velm[curlev][ie] = velm[curlev][i1];
        velm[curlev][i1] = e1;
      }
  }

CFRnumel[curlev] = i1 - FRnumel[curlev];
RURnumel[curlev] = URnumel[curlev] - CFRnumel[curlev];
```

At the end of the algorithm we also obtain the number of refinable elements that are refined *(CFRnumel)* , and the number of refinable elements that remain unchanged *(RURnumel)* .

3.5 Classification of Sides of Current Level

For each level we have a list of sides of that level. Since we are not considering derefinements of the mesh, the size of this list can only growth or remain unchanged after each refinement. New sides are added to end of this list as the same time as they are

generated. Sides are numbered independently in each level in correspondence with their position in the list of each level, we refer to this enumeration as the *'original'* enumeration of sides.

The sides of each level can be grouped into refinable and unrefinable sides. A side is considered unrefinable if its parent side is a transition side, that is the adjacent elements of this side are in different levels. The number of refinable and unrefinable sides of level ***curlev*** are stored in the next variables:

Rsides[curlev] : *refinable sides of level* ***curlev***.
Usides[curlev] : *unrefinable sides of level* ***curlev***.

The sides of the group of refinable sides are grouped into refined and unrefined sides. The number of refined and unrefined sides of the group of refinables sides of the level ***curlev*** are stored in the next variables:

FRsides[curlev] : *refined sides of level* ***curlev***.
URsides[curlev] : *refinable sides of level* ***curlev*** *not refined.*

In order to facilitate the generation of the mesh arrays of the next level we rearrange the sides of each level grouping first the refined sides, then the refinable sides and then the unrefinable ones. The new ordering of sides of each level, is stored in the integer matrix **vside**[maxlev][sides]. Then **vside**[curlev][is] stores the original identification ***os*** of the side of the level ***curlev*** identified as ***is*** in the modified enumeration. This new enumeration is schematically represented in figure 3.5, where ascending side numbering is considered from left to right.

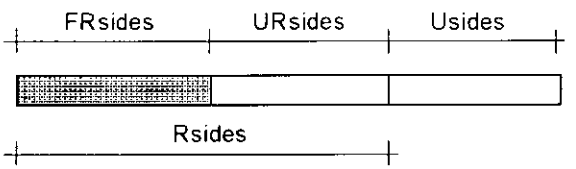


Figure 3.5. Side ordering of each level in array VSIDE [*curlev*].

Boundary conditions and boundary loading are considered applied on the sides. Thus, is convenient in each level to distinguish between boundary and internal sides. It is assumed that each boundary side is assigned to only one element, thus boundary sides are ever refinable ones. The number of boundary sides of level ***curlev*** is stored in the next variable:

Bsides[curlev] : *boundary sides of level* ***curlev***.

The boundary sides of each level are grouped into refined and unrefined boundary sides. The number of refined and unrefined boundary sides of level ***curlev*** are stored in the next variables:

BFRsides[curlev] : *refined boundary sides of level* ***curlev***.
BURsides[curlev] : *unrefined boundary sides of level* ***curlev***.

The list of boundary sides of each level, is stored in the integer matrix **bside**[maxlev][Bsides]. Where **bside**[curlev][bs] stores the original identification of the boundary side ***bs*** of the level ***curlev***.

In each level we order the boundary sides numbering first the refined sides and then the unrefined ones. This is schematically represented in figure 3.6, where ascending side numbering is considered from left to right. First we number the refinable boundary sides *(BFRsides)* and then the refinable ones *(BURsides)*.

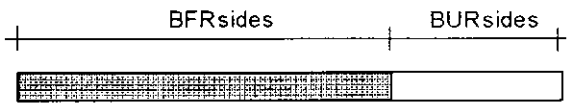


Figure 3.6. Boundary side ordering of each level in array BSIDE [*curlev*].

Also, it is convenient in each level to distinguish the transition sides to make an efficient treatment of the constrained variables of these sides. The number of transition sides of level ***curlev*** is stored in the next variable:

PRsides[curlev] : *transition sides of level curlev.*

The list of transition sides of each level, is stored in the integer matrix **tside**[maxlev][PRsides]. Where **tside**[curlev][ts] stores the original identification of the transition side ***ts*** of the level ***curlev***.

For the initial level all the sides are considered refinable, so before the first refinement we have:

Rsides[0] = **sides**[0]
FRsides[0] = 0

URsides[0] = **sides**[0]
Usides[0] = 0
BFRsides[0] = 0
BURsides[0] = **Bsides**[0]

3.6 Modification of Side Ordering After Refinement of the Current Level

If some elements of level ***curlev*** will be refined, this implies that some refinable sides of this level will be refined too, other will be transformed in transition sides and the others will remain unchanged.

Between the refined sides we must distinguish transition sides with constrained variables and active sides that have no constraints.To identify which refinable sides will be transformed in active sides, which will be transformed in transition sides and which will remain unchanged we define an integer array **eside** for each level in analysis. For the side ***os*** of the level ***curlev*** this vector can assume one of the next values:

eside[curlev][os] = 1 for side os to be transformed in active side
eside[curlev][os] = -1 for side os to be transformed in transition side
eside[curlev][os] = 0 for side os to remain unrefined

To form the vector eside for the level curlev we employ the next algorithm:

```
for (os=1; os<=sides[curlev] ; ie++) eside[curlev][os]=0;

for (ie=1; ie<=numel[curlev] ; ie++)
  {
   if (relm[curlev][ie])
    {
     L1 = abs(elm_sides[curlev][ie][1]);
     L2 = abs(elm_sides[curlev][ie][2]);
     L3 = abs(elm_sides[curlev][ie][3]);

     eside[curlev][L1] += 1;
     eside[curlev][L2] += 1;
     eside[curlev][L3] += 1;
    }
  }


for (bs=1; bs<=Bsides[curlev] ; bs++)
    if (eside[curlev][bside[curlev][bs]] == 1)
                  eside[curlev] [bsides[curlev][bs]]=2;

for (os=1; os<=sides[curlev] ; os++)
    if (eside[curlev][os] == 1) eside[curlev][os]= -1;

for (os=1; os<=sides[curlev] ; os++)
    if (eside[curlev][os] == 2) eside[curlev][os]= 1;
```

All the new refined sides pertains to the group of refinable sides. To store the number of refinable sides that are transformed into active sides, the number of refinable sides that are transformed into transition sides and the number of refinable sides that remain unrefined in each level ***curlev*** we define the next variables:

CFRsides[curlev] : *refinable sides transformed into active sides in level* ***curlev***.
CPRsides[curlev] : *refinable sides transformed into transition sides in level* ***curlev***.
RURsides[curlev] : *refinable sides that remain unrefined in level* ***curlev***.

In order to facilitate the creation of the mesh arrays of the next level we rearrange the sides of the recently refined level as indicated in Figure 3.7. We number first the existing refined sides *(FRsides)*, then the active sides *(CFRsides)* , then the refinable sides that transform into transition sides *(CPRsides)*, then the refinable sides that remain unrefined *(RURsides)*, and finally we left unchanged the unrefinable sides of the level *(Usides)*.

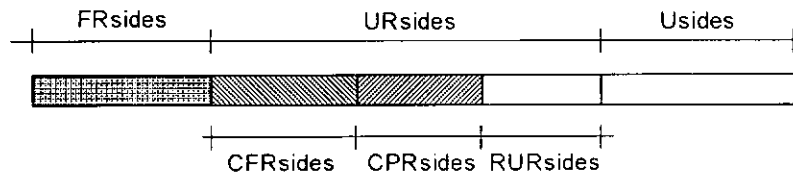


Figure 3.7. Side ordering after refinement in array VSIDE [*curlev*].

We must modify the integer matrix **vside**[curlev][sides] to store the new ordering of sides. To obtain the new ordering for the current level ***(curlev)*** we must analize the refinable sides of each level. First we order the new active sides, for do this we employ the next algorithm:

```
n1 = 0;
i1 = FRsides[curlev];

for (is=i1+1; is<=Rsides[curlev] ; is++)
  {
   if (eside[curlev][vside[curlev][is]] == 1)
    {
     ++i1;
     ++n1;
     s1 = vside[curlev][is];
     vside[curlev][is] = vside[curlev][i1];
     vside[curlev][i1] = s1;
    }
}
```

Then we order the new transition sides employing the next algorithm:

```
n2 = 0;
for (is=i1+1; is<=Rsides[curlev] ; is++)
  {
   if (eside[curlev][vside[curlev][is]] == -1)
    {
     ++i1;
     ++n2;
```

```
            s1 = vside[curlev][is];
            vside[curlev][is] = vside[curlev][i1];
            vside[curlev][i1] = s1;
          }
      }

      CFRsides[curlev] = n1;
      CPRsides[curlev] = n2;
      RURsides[curlev] = URsides[curlev] - n1 - n2;
```

At the end of the algorithm we also obtain the number of new active sides *(CFRsides)*, the number of new transition sides *(CPRsides)* and the number of refinable sides that remain unchanged *(RURsides)*.

To store the number of boundary sides that are refined in each level and the number of boundary sides that remain unrefined in each level ***curlev*** we define the next variables:

CBFRsides[curlev] : *boundary sides that are refined in level* ***curlev***.
RBURsides[curlev] : *boundary sides that remain unrefined in level* ***curlev***.

In order to facilitate the treatment of boundary sides of the next level we rearrange the boundary sides of the recently refined level as indicated in Figure 3.8. We number first the existing refined boundary sides *(BFRsides)*, then the boundary sides that must be refined *(CBFRsides)*, and finally the boundary sides that remain unrefined *(RBURsides)*.

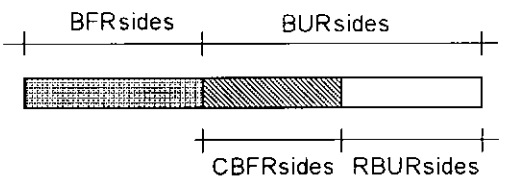


Figure 3.8. Boundary side ordering after refinement in array BSIDE [*curlev*].

We must modify the integer matrix **bside**[curlev][sides] to store the new ordering of boundary sides. To obtain the new ordering for the current level ***(curlev)*** we must analize the unrefined boundary sides of each level, to do this we employ the next algorithm:

```
i1 = BFRsides[curlev];
n1 = 0;

for (bs=i1+1 ; bs<=Bsides[curlev] ; bs++)
  {
    if (eside[curlev][bside[curlev][bs]] == 1)
      {
        ++i1;
```

```
            ++n1;
            s1 = bside[curlev][bs];
            bside[curlev][bs] = bside[curlev][i1];
            bside[curlev][i1] = s1;
          }
        }

      CBFRsides[curlev] = n1 - BFRsides[curlev];
      RBURsides[curlev] = BURsides[curlev] - n1;
```

At the end of the algorithm we also obtain the number of boundary sides that are refined *(CBFRsides)*, and the number of boundary sides that remain unrefined *(RBURsides)*.

It must be noted that the number of transition sides can be greater or less than before the refinement, since some previous transition sides can be transformed in normal refined sides if its adjacent elements are both refined. To store the number of new transition sides in each level ***curlev*** we define the next variables:

NPRsides[curlev] : *total number of transition sides after refinement in level* ***curlev***.

Then we must compute if the number of existing transition sides have changed by looping over all the refined sides using the next algorithm:

```
n1 = 0;
is1 = FRsides[curlev];
for (is=1; is<=is1 ; is++)
    if (eside[vside[curlev][is]] == -1) ++n1;

NPRsides[curlev] = n1+ CPRsides[curlev];
```

3.7 Number of Nodes, Sides and Elements Generated in the Next Level

After the refinement of each level new elements, sides and nodes are generated. The number of new generated elements, sides and nodes of each level ***curlev+1*** are stored in the next variables:

Nnumel[curlev+1] : *number of new elements generated in level* ***curlev+1***.
Nsides[curlev+1] : *number of new sides generated in level* ***curlev+1***.
Nnodes[curlev+1] : *number of new nodes generated in level* ***curlev+1***.

For linear triangles each new subdivided element *(CFRnumel)* generates four elements and three interior sides, and each new subdivided side *(CFRsides and CPRsides)* generates two sides and one node. Then if we know the number of new refined elements *(CFRnumel)*, the number of new refined sides *(CFRsides)* of the current level ***curlev*** , we can compute the number of new generated elements, sides and nodes of the next level ***curlev+1*** :

Nnumel[curlev+1] = 4 **CFRnumel**[curlev]
Nsides[curlev+1] = 2 **CFRsides**[curlev] + 2 **CPRsides**[curlev] + 3 **CFRnumel**[curlev]
Nnodes[curlev+1] = **CFRsides**[curlev] + **CPRsides**[curlev]

We can determine also the number of refinable, unrefinable and boundary sides generated in the next level ***curlev+1***, which are stored in the next variables:

NURsides[curlev+1] : *number of new refinable sides generated in level* ***curlev+1***.
NUsides[curlev+1] : *number of new unrefinable sides generated in level* ***curlev+1***.
NBsides[curlev+1] : *number of new boundary sides generated in level* ***curlev+1***.

Noting that the new transition sides *(CPRsides)* of level ***curlev*** can not be subdivided more than once, their subdivision generates only unrefinable sides *(NUsides)* on the next level ***curlev+1***. Then we have for linear triangles:

NURsides[curlev+1] = 2 **CFRsides**[curlev] + 3 **CFRnumel**[curlev]
NUsides[curlev+1] = 2 **CPRsides**[curlev]
NBsides[curlev+1] = 2 **CBFRsides**[curlev]

An element of level ***curlev+1*** is considered unrefinable if its has an adjacent transition element of level ***curlev***, thus if a refined element of level ***curlev*** has transitions sides, indicating the presence of adjacent transition elements, some of its generated elements of level ***curlev+1*** will be unrefinable. To store the number of generated refinable and unrefinable elements of each level ***curlev*** we define the next variables:

NURnumel[curlev] : *number of new refinable elements generated in level* ***curlev***.
NUnumel[curlev] : *number of new unrefinable elements generated in level* ***curlev***.

Then for the determination of the refinable and unrefinable elements of the next level ***curlev+1***, we must do an element by element analysis on the refined elements of level ***curlev***. A refined element of level ***curlev*** with one transition side will generate two unrefinable elements in level ***curlev+1***, if it has two transition sides or more it will generate three unrefinable elements and if it has no transition sides then the four generated elements will be refinables (Fig. 3.9).

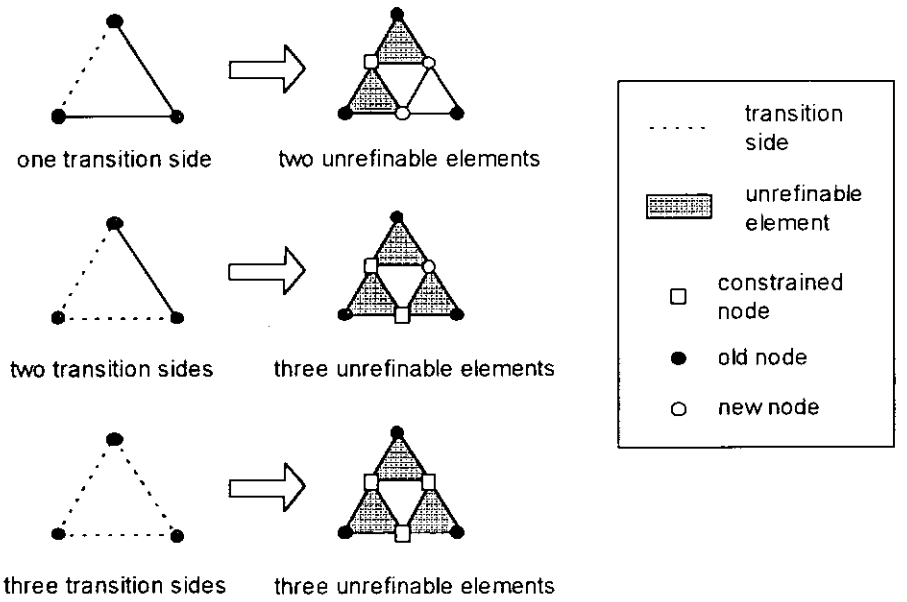


Figure 3.9. Unrefinable elements of the new level

The refined elements of each level can have normal refined sides or transition sides, where the type of each side L_i is stored in the vector **eside**[L_i]. We define an auxiliar variable ***sum*** for each refined element of the current level *(curlev)* which stores the sum of the values of the vector **eside** for all its sides, that is:

$$\textbf{\textit{sum}} = \textbf{eside}[L_1] + \textbf{eside}[L_2] + \textbf{eside}[L_3] \tag{13}$$

where L_1, L_2, L_3, are the side numbers of a refined element of the current level *(curlev)*. Taking into account that there is a unique correspondence between this sum and the number of transition sides, as showed in table 1, then we can use the value ***sum*** to obtain the number of unrefinable elements *(CURnumel)* of the level ***curlev+1*** generated during the refinement of the current level ***curlev***.

Table 3.1. Relationship between *sum* and generated unrefinable elements

sum	number of transition sides ***curlev***	number of generated unrefinable elements ***curlev+1***
3	0	0
1	1	2
-1	2	3
-3	3	3

Calling ***nel*** to the number of unrefinable elements generated during the refinement of each element of level ***curlev***, then we can express the relationship between the variables ***nel*** and ***sum*** by the next formula:

$$nel = (3 - sum)(sum + 7)/8 \qquad (14)$$

Then we employ the next algorithm to obtain the number of new generated refinable and unrefinable elements of the new level *(**curlev+1**)* :

```
n1 = 0;
ie1 = FRnumel[curlev];
ie2 = ie1 + CFRnumel[curlev];
for (ie= ie1+1 ; ie<=ie2 ; ie++)
  {
    L1 = abs(elm_sides[curlev][velm[curlev][ie]][1]);
    L2 = abs(elm_sides[curlev][velm[curlev][ie]][2]);
    L3 = abs(elm_sides[curlev][velm[curlev][ie]][3]);

    sum = eside[curlev][L1] + eside[curlev][L2] + eside[curlev][L3];
    n1 += (3-sum)*(sum+7)/8;
  }
 }

NUnumel[curlev+1] = n1;
NURnumel[curlev+1] = Nnumel[curlev+1] - n1;
```

At the end of the algorithm we have obtained the number of new generated refinable *(NURnumel)* and unrefinable *(NUnumel)* elements of the next level ***curlev+1***.

3.8 Modification of Side Ordering of the Next Level.

Assuming that the level ***curlev+1*** was generated by a previous refinement, and taking into account that an unrefinable side of the level ***curlev+1*** is generated by the subdivision of a transition side of the current level ***curlev*** , then it is possible that if some previous transition side of the level ***curlev*** change to normal refined then some unrefinable sides of the level ***curlev+1*** change into refinable ones after the refinement of the current level ***curlev***. To store the number of unrefinable sides that are transformed into refinable and the number of unrefinable sides that remain unrefinable in the next level ***curlev+1*** we define the next variables:

CURsides[curlev+1] : *unrefinable sides transformed into refinable in level **curlev+1**.*

RUsides[curlev+1] : *unrefinable sides that remain unrefinable in level **curlev+1**.*

Then we rearrange the unrefinable sides *(Usides)* of the next level ***curlev+1*** as indicated in Figure 3.10. We number first the transformed sides *(CURsides)* and then the unrefinable sides that remain unrefinable *(RUsides)*.

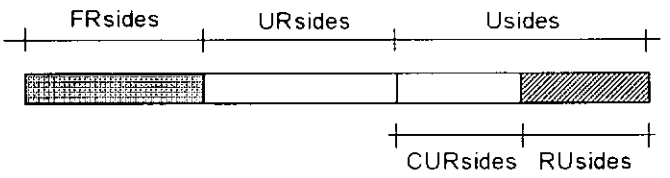


Figure 3.10. Unrefinable side re-ordering in array VSIDE [*curlev+1*].

We must inspect the transition sides before the refinement of the current level ***curlev***, if these sides remain as transition sides after the refinement then their generated sides in the next level ***curlev+1*** remain unrefinable. To identify the new order of sides it is convenient to know the inverse side ordering of the next level ***curlev+1***, that is the new order corresponding to an original side of the list of sides of that level. Instead of defining a new array for the new enumeration of sides we employ the array **eside** which is needed no more after the modifications of each level have been made. Then the components of the array **eside**[curlev+1] of the next level satisfy the next relation

eside[curlev+1][vside[curlev+1][is]] = is;

Also we define the integer matrix **ssside**[maxlev][sides]. Where **ssside**[curlev][os] stores the first of the two consecutive sides generated in the next level ***curlev+1*** by subdivisition of the original side ***os*** of the level ***curlev***. Then the unrefinables sides of the level ***curlev+1*** are reordered using the next algorithm:

```
n1 = Rsides[curlev+1];
i1 = n1;

for (is=1; is<= PRsides[curlev]; is++)
  {
    if (eside[curlev][tside[curlev][is]] != -1)
     {
       side = ssside[curlev] [tside[curlev][is]];
       side1 = eside[level+1][side];
       side2 = eside[level+1][side+1];

       ++i1;
       s1 = vside[curlev+1][side1];
       s2 = vside[curlev+1][i1];
       vside[curlev+1][side1] = s2;
       vside[curlev+1][i1] = s1;
       vside[curlev+1][s1] = i1;
       vside[curlev+1][s2] = side1;

       ++i1;
       s1 = vside[curlev+1][side2];
       s2 = vside[curlev+1][i1];
       vside[curlev+1][side2] = s2;
       vside[curlev+1][i1] = s1;
```

```
            vside[curlev+1][s1] = i1;
            vside[curlev+1][s2] = side2;
          }
        }

      CURsides[curlev+1] = i1 - n1;
      RUsides[curlev+1] = Usides[curlev+1] - CURsides[curlev+1];

      URsides[curlev+1] += CURsides[curlev+1];
      Usides[curlev+1] = RUsides[curlev+1];
      Rsides[curlev+1] += CURsides[curlev+1];
```

At the end of the algorithm we also update the number of refinable sides *(Rsides)*, unrefinable sides *(Usides)* sides and refinable sides not refined *(URsides)* of the next level ***curlev+1***.

Finally we must fill the integer matrix **tside**[curlev][NPRsides] to identify the new transition sides of the current level ***curlev***. To do this we employ the next algorithm:

```
n1 = 0;
for (is=1; is<= FRsides[curlev] ; is++)
    if (eside[curlev][vside[curlev][is]] == -1)
                tside[curlev][++n1]= vside[curlev][is];

is1 = FRsides[curlev]  + CFRsides[curlev];
is2 = is1 + CPRsides[curlev];
for (is=is1+1; is<=is2 ; is++)  tside[curlev][++n1]= vside[curlev][is];
```

Since all modifications to the ordering of sides of the level ***curlev*** were made, then the array **eside**[curlev] can be used to store the inverse side ordering of the current level ***curlev***. To do this we employ the next algorithm:

```
for (is=1 ; is<=sides[curlev] ; is++)
            eside[curlev][ vside[curlev][is] ] = is;
```

3.9 Modification of Element Ordering of the Next Level.

Assuming that the level ***curlev+1*** was generated by a previous refinement, and taking into account that some previous transition sides of the level ***curlev*** can change to normal refined sides, then some unrefinables elements of the level ***curlev+1*** can change into refinables ones after the refinement of the current level ***curlev***. To store the number of unrefinable elements that are transformed into refinable and the number of unrefinable elements that remain unrefinable in the next level ***curlev+1*** we define the next variables:

CURnumel[curlev+1] : *unrefinable elements changed to refinable in level* ***curlev+1***.
RUnumel[curlev+1] : *unrefinable elements that remain unrefinable in level* ***curlev+1***.

Then we rearrange the unrefinable elements *(Unumel)* of the next level ***curlev+1*** as indicated in Figure 3.11. We number first the transformed elements *(CURnumel)* and then the unrefinable sides that remain unrefinable *(RUnumel)*.

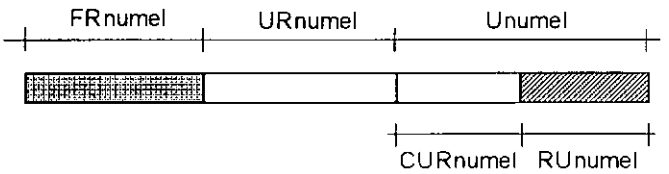


Figure 3.11. Unrefinable element re-ordering in array VELM [*curlev+1*].

We must inspect the element sides of the unrefinable elements of the next level ***curlev+1***, if some of these sides remain as unrefinable sides after the refinement then the element remain unrefinable. To determine the number of unrefinable and refinable elements we employ the next algorithm:

```
s1 = Rsides[curlev+1];
n1 = Rnumel[curlev+1];
i1 = n1;

for (ie=i1+1; ie<= numel[curlev+1]; ie++)
  {
     e1 = velm[curlev+1][ie];

     L1 = abs(elm_sides[curlev+1][e1][1]);
     L2 = abs(elm_sides[curlev+1][e1][2]);
     L3 = abs(elm_sides[curlev+1][e1][3]);

     NL1 = eside[curlev+1][L1];
     NL2 = eside[curlev+1][L2];
     NL3 = eside[curlev+1][L3];

     if ( (NL1<= s1)&& (NL2<= s1)&& (NL3<= s1) )
      {
        ++i1;
        velm[curlev+1][ie] = velm[curlev+1][i1];
        velm[curlev+1][i1] = e1;
      }
  }

CURnumel[curlev+1] = i1 - n1;
RUnumel[curlev+1] = Unumel[curlev+1] - CURnumel[curlev+1];

URnumel[curlev+1] += CURnumel[curlev+1];
Unumel[curlev+1]   = RUnumel[curlev+1];
Rnumel[curlev+1] += CURnumel[curlev+1];
```

At the end of the algorithm we also update the number of refinable elements *(Rnumel)* , unrefinable elements *(Unumel)* , and refinable elements not refined *(URnumel)* of the next level ***curlev+1***.

3.10 Side and Node Generation by Subdivision of Sides

In the level ***curlev+1*** we have three types of generated sides:

- Refinable sides generated by the subdivision of normal sides of the current level *(CFRsides[curlev])*.
- Refinable interior sides generated by the subdivision of elements of the current level *(CFRnumel[curlev])*.
- Unrefinable sides generated by the subdivision of transition sides of the current level *(CPRsides[curlev])*.

Using the side ordering after the refinement of the level ***curlev*** is possible to order directly the sides of the new level ***curlev+1*** in refinable and unrefinable groups by numbering first the sides generated by subdivision of normal sides, then those generated by subdivision of elements and finally the sides generated by subdivision of transition sides (Fig. 3.12).

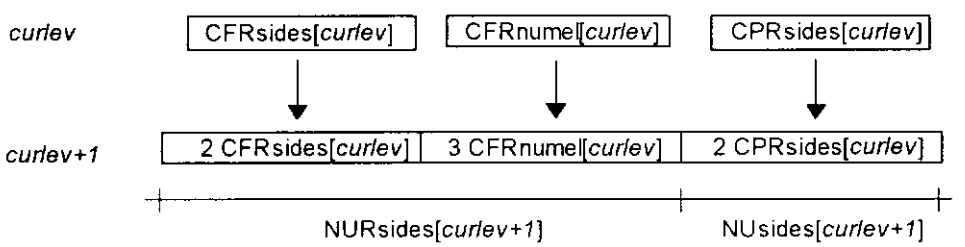


Figure 3.12. Side generation by refinement of triangular elements

Then we must expand the vector **vside**[curlev+1] to insert the new generated sides of the level ***curlev+1*** as indicated in Figure 3.13. We number first the new refinable sides *(NURsides)* and then the unrefinable sides *(NUsides)*.

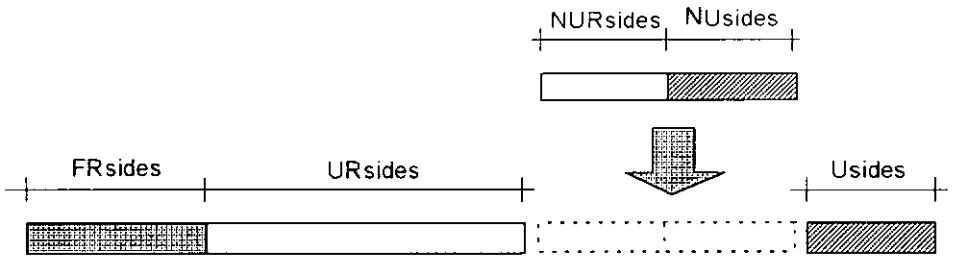


Figure 3.13. Expansion and insertion of new sides in array VSIDE [*curlev+1*].

Then we employ the next algorithm for fill the array **vside**[curlev+1] with the identification of the new sides which are added to this level:

```
n1 = sides[curlev+1];
i1 = Rsides[curlev+1];

is1 = i1+1;
is2 = i1+Nsides[curlev+1];
for (is=is1; is<=is2 ; is++) vside[curlev+1][is] = ++n1;
for (is=1; is<=is2 ; is++) eside[curlev+1][ vside[curlev+1][is] ] = is;
```

It must be noted that the array **eside**[curlev+1] is modified accordingly to store the new inverse enumeration.

Each refined parent side *(CFRsides and CPRsides)* of the current level ***(curlev)*** generates two son sides and one node in the next level ***(curlev+1)*** (Fig. 3.14).

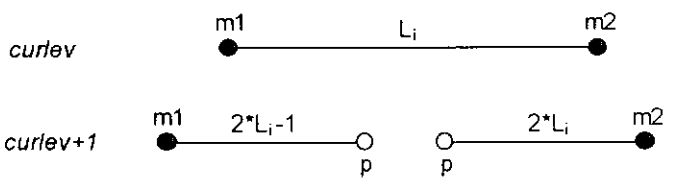


Figure 3.14. Side and node generation by side refinement.

Then we must fill the array **ssside**[curlev] of the current level ***curlev*** using the next algorithm:

```
n1 = FRsides[curlev];
n2 = sides[curlev+1];
for (is=1 ; is<=CFRsides[curlev] ; is++)
        ssside[curlev][ vside[curlev][n1+is] ] = n2+2*is-1;

n1 = Frsides[curlev] + CFRsides[curlev];
n2 = sides[curlev+1] + NURsides[curlev+1];
```

```
for (is=1 ; is<=CPRsides[curlev] ;  is++)
        ssside[curlev][ vside[curlev][n1+is] = n2+2*is-1;
```

To assign the incidences of the refinable sides of the next level *(curlev+1)*, which are generated by subdivision of normal sides of the current level *(curlev)* we employ the next algorithm:

```
n1 = nodes[curlev+1];
is1 = FRsides[curlev];
is2 = is1 + CFRsides[curlev];
for (is=is1+1 ; is<=is2 ;  is++)
      {
        m1 = side_nodes[curlev][ vside[curlev][is] ][1];
        m2 = side_nodes[curlev][ vside[curlev][is] ][2];
        lv1 = side_vlev[curlev][ vside[curlev][is] ][1]
        lv2 = side_vlev[curlev][ vside[curlev][is] ][2];

        side1 = ssside[curlev][ vside[curlev][is] ];
        side2 = side1 + 1;

        side_nodes[curlev+1][side1][1] = m1;
        side_nodes[curlev+1][side1][2] = ++n1;
        side_nodes[curlev+1][side2][1] = n1;
        side_nodes[curlev+1][side2][2] = m2;

        side_vlev[curlev+1][side1][1] = lv1;
        side_vlev[curlev+1][side1][2] = curlev+1;
        side_vlev[curlev+1][side2][1] = curlev+1;
        side_vlev[curlev+1][side2][2] = lv2;
      }
```

To assign the incidences of the unrefinable sides of the next level *(curlev+1)*, which are generated by subdivision of transition sides of the current level *(curlev)* we employ the next algorithm:

```
is1 = Frsides[curlev] + CFRsides[curlev];
is2 = is1 + CPRsides[curlev];
for (is=is1+1 ; is<=is2 ;  is++)
      {
        m1 = side_nodes[curlev][ vside[curlev][is] ][1];
        m2 = side_nodes[curlev][ vside[curlev][is] ][2];
        lv1 = side_vlev[curlev][ vside[curlev][is] ][1]
        lv2 = side_vlev[curlev][ vside[curlev][is] ][2];

        side1 = ssside[curlev][ vside[curlev][is] ];
        side2 = side1 + 1;

        side_nodes[curlev+1][side1][1] = m1;
        side_nodes[curlev+1][side1][2] = ++n1;
        side_nodes[curlev+1][side2][1] = n1;
```

```
    side_nodes[curlev+1][side2][2] = m2;

    side_vlev[curlev+1][side1][1] = lv1;
    side_vlev[curlev+1][side1][2] = curlev+1;
    side_vlev[curlev+1][side2][1] = curlev+1;
    side_vlev[curlev+1][side2][2] = lv2;
  }
```

We must also expand the vector **bside**[curlev+1] to insert the new generated boundary sides of the level ***curlev+1*** as indicated in Figure 3.15. These new sides are generated by the subdivision of the refined boundary sides *(CBFRsides)* of the previous level ***curlev***.

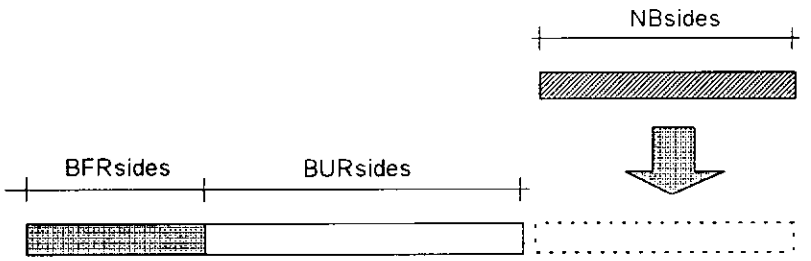


Figure 3.15. Expansion and addition of new sides in array BSIDE |*curlev+1*|.

Then we employ the next algorithm for fill the array **bside** with the identification of the new boundary sides of this level:

```
is1 = BFRsides[curlev] + 1;
is2 = BFRsides[curlev] + CBFRsides[curlev];
for (is=is1; is<=is2 ; is++)
  {
    s1 = ssside[curlev][bside[curlev][is]];

    bside[curlev+1][2*is-1] = s1;
    bside[curlev+1][2*is] = s1+1;
  }
```

For each refined side we have a new intermediate node. In the case of straight sides the coordinates of these new nodes are computed as the mean values of the side end coordinates. Then we use the next algorithm to compute the coordinates of the new nodes:

```
is1 = nodes[curlev+1];
is2 = is1 + Nnodes[curlev+1];
for (is=is1+1 ; is<=is2 ; is++)
  {
    L1 = vside[curlev][is];
```

```
        n1 = side_nodes[curlev][L1][1]
        n2 = side_nodes[curlev][L1][2];

        lv1 = side_vlev[curlev][L1][1]
        lv2 = side_vlev[curlev][L1][2];

        x1 = x[lv1][n1];
        y1 = y[lv1][n1];
        x2 = x[lv2][n2];
        y2 = y[lv2][n2];

        x[curlev+1][is] = (x1+x2)/2;
        y[curlev+1][is] = (y1+y2)/2
}
```

3.11 Element and Side Generation by Subdivision of Elements

In the level ***curlev+1*** we have refinable *(NURnumel[curlev+1])* and unrefinable *(NUnumel[curlev+1])* elements generated by the subdivision of refinable elements *(CFRnumel[curlev])* of the previous level ***curlev***.

Then we must expand the vector **velm**[curlev+1] to insert the new generated elements of the level ***curlev+1*** as indicated in Figure 3.16. We number first the new refinable elements *(NURnumel)* and then the unrefinable elements *(NUnumel)*.

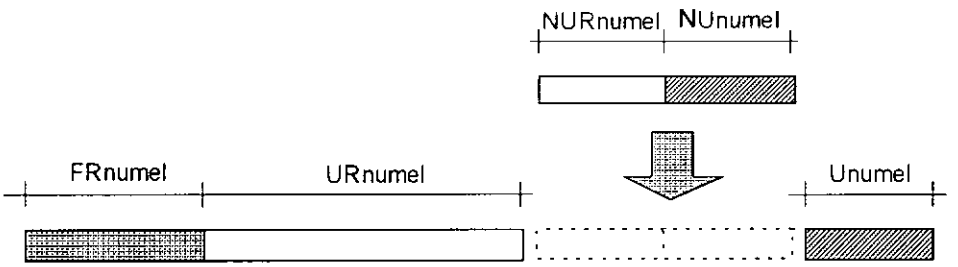


Figure 3.16. Expansion and insertion of new sides in array VELM [*curlev+1*].

Each subdivided element of the current level ***(curlev)*** generates four elements (e_1, e_2, e_3, e_4) and three interior sides (s_1, s_2, s_3) in the new level ***(curlev+1)*** (Fig. 3.17). The other sides located on the element boundary (s_4, s_5, s_6, s_7, s_8, s_9) and the midside nodes (m_1, m_2, m_3) are considered generated by the subdivision of sides.

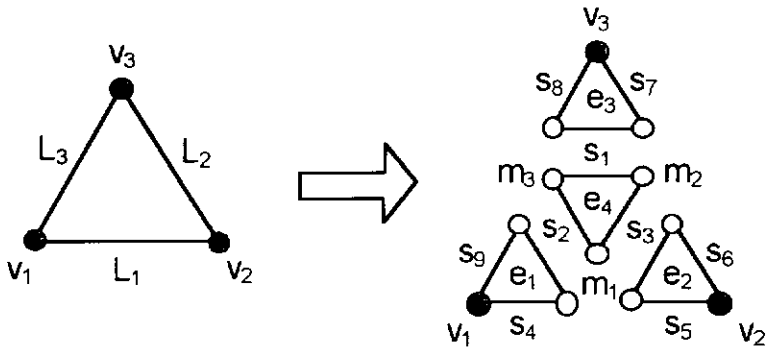


Figure 3.17. Triangular element refinement

We define the integer matrix **seside**[maxlev][numel]. Where **seside**[curlev][oe] stores the first of the three consecutive interior sides s1, s2, s3, generated in the next level ***curlev+1*** by subdivisition of the original element ***oe*** of the level ***curlev***. Then we must fill the array **seside**[curlev] of the current level ***curlev*** using the next algorithm:

```
n1 = FRnumel[curlev];
n2 = sides[curlev+1] + 2*CFRsides[curlev];
for (is=1 ; is<=CFRnumel[curlev] ;  is++)
        seside[curlev][ velm[curlev][n1+is] ] = n2+3*is-2;
```

Then we can employ the next algorithm to assign the sides to the elements of the new level *(**curlev+1**)* and to assign the incidences of the interior sides generated by subdivision of elements of the previous level *(**curlev**)*:

```
/*************************************************************/
/*                  auxiliar parameters                      */
/*************************************************************/

n1 = 2*(FRsides[curlev] + CFRsides[curlev]);
n4 = Rnumel[curlev+1];
n3 = n4 + NURnumel[curlev+1];
n6 = numel[curlev+1];
n7 = FRsides[curlev] + CFRsides[curlev];

/*************************************************************/
/*       Loop on elements to be refined                      */
/*************************************************************/

ie1 = FRnumel[curlev];
ie2 = ie1 + CFRnumel[curlev];

for (ie=ie1+1 ; ie<=ie2 ; ie++)
   {

/*****************************************************************/
```

```
/*  sides generated by element boundary subdivision  */
/*****************************************************************/

    L1 = elm_sides[curlev][ velm[curlev][ie] ][1];
    L2 = elm_sides[curlev][ velm[curlev][ie] ][2];
    L3 = elm_sides[curlev][ velm[curlev][ie] ][3];

    NL1 = ssside[curlev][abs(L1)];
    NL2 = ssside[curlev][abs(L2)];
    NL3 = ssside[curlev][abs(L3)];

    if (L1<0)
     {
       s4 = -NL1-1;
       s5 = -NL1;
     }
    else
     {
       s4 = NL1;
       s5 = NL1+1;
     }

    if (L2<0)
     {
       s6 = -NL2-1;
       s7 = -NL2;
     }
    else
     {
       s6 = NL2;
       s7 = NL2+1;
     }

    if (L3<0)
     {
       s8 = -NL3-1;
       s9 = -NL3;
     }
    else
     {
       s8 = NL3;
       s9 = NL3+1;
     }


/*********************************************************/
/*        vertex nodes of parent triangle                */
/*********************************************************/

v1= elm_nodes[curlev][ velm[curlev][ie] ][1];
```

```
v2= elm_nodes[curlev][ velm[curlev][ie] ][2];
v3= elm_nodes[curlev][ velm[curlev][ie] ][3];

/*************************************************************/
/*          vertex levels of parent triangle                 */
/*************************************************************/

lv1= elm_vlev[curlev][ velm[curlev][ie] ][1];
lv2= elm_vlev[curlev][ velm[curlev][ie] ][2];
lv3= elm_vlev[curlev][ velm[curlev][ie] ][3];

/*************************************************************/
/* midside nodes generated on parent triangle       */
/*************************************************************/

m1 = eside[curlev][abs(L1)];
m2 = eside[curlev][abs(L2)];
m3 = eside[curlev][abs(L3)];

/*****************************************************************/
/*internal sides generated by element subdivision          */
/*****************************************************************/

s1 = seside[curlev][ velm[curlev][ie] ];
s2 = s1 + 1;
s3 = s2 + 1;

side_nodes[curlev+1][s1][1] = m2;
side_nodes[curlev+1][s1][2] = m3;
side_nodes[curlev+1][s2][1] = m3;
side_nodes[curlev+1][s2][2] = m1;
side_nodes[curlev+1][s3][1] = m1;
side_nodes[curlev+1][s3][2] = m2;

side_vlev[curlev+1][s1][1] = curlev+1;
side_vlev[curlev+1][s1][2] = curlev+1;
side_vlev[curlev+1][s2][1] = curlev+1;
side_vlev[curlev+1][s2][2] = curlev+1;
side_vlve[curlev+1][s3][1] = curlev+1;
side_vlev[curlev+1][s3][2] = curlev+1;

/*****************************************************************/
/* new elements generated by element subdivision       */
/*****************************************************************/

e1 = ++n6;
e2 = ++n6;
e3 = ++n6;
e4 = ++n6;
```

```
if (m1>n7  ||  m3>n7)
     velm[curlev+1][++n3] = e1;
else
     velm[curlev+1][++n4] = e1;

if (m1>n7  || m2>n7)
     velm[curlev+1][++n3] = e2;
else
     velm[curlev+1][++n4] = e2;

if (m2>n7 ||  m3>n7)
     velm[curlev+1][++n3] = e3;
else
     velm[curlev+1][++n4] = e3;

velm[curlev+1][++n4] = e4;

/*****************************************************************/
/* side assignment of new elements                               */
/*****************************************************************/

elm_sides[curlev+1][e1][1] =  s4;
elm_sides[curlev+1][e1][2] = -s2;
elm_sides[curlev+1][e1][3] =  s9;

elm_sides[curlev+1][e2][1] =  s5;
elm_sides[curlev+1][e2][2] =  s6;
elm_sides[curlev+1][e2][3] = -s3;

elm_sides[curlev+1][e3][1] = -s1;
elm_sides[curlev+1][e3][2] =  s7;
elm_sides[curlev+1][e3][3] =  s8;

elm_sides[curlev+1][e4][1] =  s1;
elm_sides[curlev+1][e4][2] =  s2;
elm_sides[curlev+1][e4][3] =  s3;

/*****************************************************************/
/* nodal incidences of new elements                              */
/*****************************************************************/

elm_nodes[curlev+1][e1][1] =  v1;
elm_nodes[curlev+1][e1][2] =  m1;
elm_nodes[curlev+1][e1][3] =  m3;

elm_nodes[curlev+1][e2][1] =  m1;
elm_nodes[curlev+1][e2][2] =  v2;
elm_nodes[curlev+1][e2][3] =  m2;

elm_nodes[curlev+1][e3][1] =  m3;
```

```
elm_nodes[curlev+1][e3][2] =  m2;
elm_nodes[curlev+1][e3][3] =  v3;

elm_nodes[curlev+1][e4][1] =  m2;
elm_nodes[curlev+1][e4][2] =  m3;
elm_nodes[curlev+1][e4][3] =  m1;

/*****************************************************************/
/* level of new element vertexs                                 */
/*****************************************************************/

elm_vlev[curlev+1][e1][1] =  lv1;
elm_vlev[curlev+1][e1][2] =  curlev+1;
elm_vlev[curlev+1][e1][3] =  curlev+1;

elm_vlev[curlev+1][e2][1] =  curlev+1;
elm_vlev[curlev+1][e2][2] =  lv2;
elm_vlev[curlev+1][e2][3] =  curlev+1;

elm_vlev[curlev+1][e3][1] =  curlev+1;
elm_vlev[curlev+1][e3][2] =  curlev+1;
elm_vlev[curlev+1][e3][3] =  lv3;

elm_vlev[curlev+1][e4][1] =  curlev+1;
elm_vlev[curlev+1][e4][2] =  curlev+1;
elm_vlev[curlev+1][e4][3] =  curlev+1;

/*************************************************************/
/* end of loop on elements to be refinaned                   */
/*************************************************************/

}
```

3.12 Updating Mesh Parameters After Refinement

We must update the new total number of refined and refinable sides and elements of the current level ***curlev*** , as:

```
FRnumel [curlev] += CFRnumel [curlev];
URnumel [curlev] = RURnumel [curlev];

FRsides [curlev] += CFRsides [curlev] + CPRsides[curlev];
URsides [curlev] = RURsides[curlev];

BFRsides [curlev] += CBFRsides [curlev];
BURsides [curlev] = RBURsides[curlev];

Prsides[curlev] = NPRsides[curlev];
```

Finally we must update the new total number of refinable and unrefinable sides and elements of the next level ***curlev+1*** , as:

```
URnumel [curlev+1] += NURnumel [curlev+1];
Unumel [curlev+1] += NUnumel [curlev+1];
Rnumel [curlev+1] += NURnumel [curlev+1];

URsides [curlev+1] += NURsides [curlev+1];
Usides [curlev+1] += NUsides [curlev+1];
Rsides [curlev+1] += NURsides [curlev+1];

BURsides [curlev+1] += NBsides [curlev+1];

nodes[curlev+1] += Nnodes[curlev+1];
numel[curlev+1] += Nnumel[curlev+1];
sides[curlev+1] += Nsides[curlev+1];
Bsides[curlev+1] += NBsides[curlev+1];
```

Chapter 4

Multilevel Solvers

4.1. Introduction

Iterative solvers were used by the first pioneers of the finite element (FE) method in the early sixties, but they were soonly discarded when it was found that for certain problems the number of operations needed to converge were far beyond the predicted theoretical limits. Iterative solvers were then abandoned and the more effective direct methods based on triangular factorization of the matrix were adopted by structural analysts as their method of choice.

The increasing computing power reached with vector and parallel computers has renewed the interest in iterative solvers which can exploit the potential of these new architectures more efficiently than with direct solvers, especially for very large problems (more than 100,000 elements), where a direct method requires huge storage demands which is often the limiting factor to the size of problems that can be solved. On the other hand one of the most desirable advantages of iterative methods for solving large-scale linear FE equations is the low storage requirements. The coefficient matrix is preserved in its original form and is treated as a linear operator for computing matrix-vector products only, which can be done in an element-by-element parallel fashion.

In connection with adaptive analysis iterative methods are preferable since a very good approximation to the solution is known from the previous coarse mesh analysis, which can increase the speed of convergence of the iterative procedure. But, the performance of iterative methods is strongly influenced by the numerical values of the data of the problem to be solved. An important goal is thus to develop robust iterative methods that perform well for a wide range of data. Among the most efficient iterative solvers we have the *Preconditioned Conjugate Gradient (PCG)* [1], which performance depends on the type of preconditioner employed. Another family of efficient iterative algorithms are the *Multigrid methods* [63-66] which requires a hierarchy of discretizations, that is, several discretizations of increasing density for the same problem must be available. The main purpose of multigrid methods is to provide an iterative algorithm where the amount of computational work is proportional to the number of unknowns of the finer grid.

If we substitute the standard nodal bases by hierarchical bases to generate the finite element matrices when a mesh corresponding to a given level is refined, the resulting discretization preserves all the information of the previous levels. When multigrid methods are utilized with hierarchical bases, we have the *Hierarchical Basis Multigrid (HBM) method* [67]. The mathematical analysis of the HBM method [67-69] shows that it shares the same basic properties of the standard multigrid method, but under less stringent conditions.

The multilevel hierarchical preconditioners are robust preconditioners based on some approximation of the hierarchical matrices as preconditioner for PCG. This preconditioning technique has its basic theory described by Bank and Yserentant [68], Yserentant [70] for the case of symmetric positive definite matrix systems and was already tested for 3D elastostatic analysis using FE by Coutinho *et al.*[71] and for 2D elastostatic analysis using boundary elements by Barra *et al.*[72], presenting a very good performance when compared with standard preconditioners, such as nodal-block diagonal and the element-by-element Gauss-Seidel preconditioners.

4.2. The Preconditioned Conjugate Gradient method.

The conjugate gradient method was first introduced by Hestenes and Stiefel [73] in 1952 but it was not until 1970 that the method was presented as an iterative method for the solution of large sparse systems of linear equations [74]. In this section we describe the basics of the method.

4.2.1. Gradient methods.

We consider here the solution of the linear system of equations

$$\boldsymbol{K}\,\boldsymbol{x} = \boldsymbol{f} \tag{4.1}$$

where $\boldsymbol{K}$ is a $n \times n$ positive definite symmetric matrix .

Solving this linear equation is equivalent to obtain the minimum of the associated function

$$\phi(\boldsymbol{x}) = \tfrac{1}{2}\, \boldsymbol{x}^t \boldsymbol{K}\, \boldsymbol{x} - \boldsymbol{x}^t \boldsymbol{f} \tag{4.2}$$

In the gradient methods a new approximation $\boldsymbol{x}^{(k+1)}$ is obtained from $\boldsymbol{x}^{(k)}$ by the equation

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \alpha^{(k)}\, \boldsymbol{p}^{(k)} \tag{4.3}$$

which defines a line passing by the point $\boldsymbol{x}^{(k)}$ in the direction $\boldsymbol{p}^{(k)}$. After substituting in Eqn.(4.2) we have

$$\phi(\boldsymbol{x}^{(k+1)}) = \tfrac{1}{2} (\alpha^{(k)})^2 (\boldsymbol{p}^{(k)})^t \boldsymbol{K}\, \boldsymbol{p}^{(k)} - \alpha^{(k)} (\boldsymbol{p}^{(k)})^t \boldsymbol{r}^{(k)} - \tfrac{1}{2} (\boldsymbol{x}^{(k)})^t \boldsymbol{r}^{(k)} \tag{4.4}$$

where

$$\boldsymbol{r}^{(k)} = \boldsymbol{f} - \boldsymbol{K}\, \boldsymbol{x}^{(k)} \tag{4.5}$$

is the *residual* vector. Deriving Eqn.(4.4) with respect to $\alpha^{(k)}$, and equating to zero we have

$$\frac{\partial \phi}{\partial \alpha^{(k)}} = \alpha^{(k)} (p^{(k)})^t K p^{(k)} - (p^{(k)})^t r^{(k)} = 0 \qquad (4.6)$$

resulting

$$\alpha^{(k)} = (p^{(k)})^t r^{(k)} / (p^{(k)})^t K p^{(k)} \qquad (4.7)$$

The different methods differ in the direction $p^{(k)}$ choosen.

4.2.2. Steepest descent method.

In this method the direction $p^{(k)}$ is chosen as the direction of maximum gradient of the function ϕ at point $x^{(k)}$. It can be easily proved that this direction is proportional to the residual vector $r^{(k)}$, then the iterative process to obtain the solution of Eqn.(4.1) can be written as

Given:	$\mathbf{x}^{(0)}$, k = 0
Initialize:	$\mathbf{r}^{(0)} = \mathbf{f} - \mathbf{K}\,\mathbf{x}^{(0)}$
Iterate over:	$\alpha^{(k)} = \dfrac{(\mathbf{r}^{(k)}, \mathbf{r}^{(k)})}{(\mathbf{r}^{(k)}, \mathbf{K}\,\mathbf{r}^{(k)})}$
	$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)}\,\mathbf{r}^{(k)}$
	$\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha^{(k)}\,\mathbf{K}\,\mathbf{r}^{(k)}$
	k = k + 1

Figure 4.1. The Steepest Descent Method

4.2.3. Conjugated Gradient methods.

In this method the directions $p^{(k)}$ are chosen to better represent the directions of steepest descent at point $x^{(k)}$, but with the additional constraint of been mutually conjugated. Here the term conjugated means orthogonality with respect to matrix K, that is

$$(p^{(i)})^t K p^{(j)} = 0 \qquad \forall\ i \neq j \qquad (4.8)$$

The vectors $p^{(k+1)}$ can be expressed as

$$p^{(k+1)} = r^{(k)} + \beta^{(k)}\ p^{(k)} \qquad (4.9)$$

where the constant β_k is determined by the K - conjugacy property of the direction $p^{(k)}$, then the iterative process can be described as

Given: $\mathbf{x}^{(0)}$, $k = 0$

Initialize:

$$\mathbf{r}^{(0)} = \mathbf{f} - \mathbf{K}\,\mathbf{x}^{(0)}$$

$$\mathbf{p}^{(0)} = \mathbf{r}^{(0)}$$

Iterate over:

$$\alpha^{(k)} = \frac{(\mathbf{r}^{(k)}, \mathbf{p}^{(k)})}{(\mathbf{p}^{(k)}, \mathbf{K}\,\mathbf{p}^{(k)})}$$

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)}\,\mathbf{p}^{(k)}$$

$$\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha^{(k)}\,\mathbf{K}\,\mathbf{p}^{(k)}$$

$$\beta^{(k)} = \frac{(\mathbf{r}^{(k+1)}, \mathbf{K}\,\mathbf{p}^{(k)})}{(\mathbf{p}^{(k)}, \mathbf{K}\,\mathbf{p}^{(k)})}$$

$$\mathbf{p}^{(k+1)} = \mathbf{r}^{(k+1)} + \beta^{(k)}\,\mathbf{p}^{(k)}$$

$$k = k + 1$$

Figure 4.2. The Conjugate Gradient Method

It can be proved by induction that the next orthogonality conditions are satisfied

$$(\boldsymbol{r}^{(i)})^t\, \boldsymbol{p}^{(j)} = 0 \qquad \forall\, i > j \tag{4.10}$$

$$(\boldsymbol{r}^{(i)})^t\, \boldsymbol{r}^{(j)} = 0 \qquad \forall\, i \neq j \tag{4.11}$$

Due to the orthogonality conditions the correct solution would be theoretically obtained after n steps, where n is the number of equations. Therefore the method must not be classified as iterative. However, it must be noted that convergence is rarely attained in exactly n steps. More or less steps are needed. Therefore the method must be considered as iterative.

4.2.4. Preconditioning.

Preconditioning is introduced to alleviate difficulties associated with the slow rate of convergence of the conjugate gradient method. Instead of solving the original system (4.1) one solves

$$\boldsymbol{K}\,\boldsymbol{B}^{-1}\; \boldsymbol{y} = \boldsymbol{f} \tag{4.12}$$

where $\boldsymbol{x} = \boldsymbol{B}^{-1}\; \boldsymbol{y}$. The matrix $\boldsymbol{B}$ is referred to as the preconditioning matrix. The advantages of preconditioning can also be realized by solving $\boldsymbol{B}^{-1}\boldsymbol{K}\; \boldsymbol{x} = \boldsymbol{B}^{-1}\; \boldsymbol{f}$. The number of iterations required to solve (4.12) depends on the condition number, κ, of its coefficient matrix. $\kappa(\boldsymbol{K}\boldsymbol{B}^{-1})$ which is the ratio of the largest eigenvalue of the eigenproblem $(\boldsymbol{K} - \lambda\, \boldsymbol{B})\, \boldsymbol{z} = \boldsymbol{0}$ to the smallest (i.e., $\kappa = ||\boldsymbol{K}\boldsymbol{B}^{-1}|| \,/\, ||\boldsymbol{B}\boldsymbol{K}^{-1}||$).

Given: $\mathbf{x}^{(0)}$, $k = 0$

Initialize:

$$\mathbf{r}^{(0)} = \mathbf{f} - \mathbf{K}\,\mathbf{x}^{(0)}$$

$$\mathbf{z}^{(0)} = \mathbf{B}^{-1}\,\mathbf{r}^{(0)}$$

$$\mathbf{p}^{(0)} = \mathbf{z}^{(0)}$$

Iterate over:

$$\alpha^{(k)} = \frac{(\mathbf{r}^{(k)}, \mathbf{z}^{(k)})}{(\mathbf{p}^{(k)}, \mathbf{K}\,\mathbf{p}^{(k)})}$$

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)}\,\mathbf{p}^{(k)}$$

$$\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha^{(k)}\,\mathbf{K}\,\mathbf{p}^{(k)}$$

$$\mathbf{z}^{(k+1)} = \mathbf{B}^{-1}\,\mathbf{r}^{(k+1)}$$

$$\beta^{(k)} = \frac{(\mathbf{r}^{(k+1)}, \mathbf{z}^{(k+1)})}{(\mathbf{p}^{(k)}, \mathbf{K}\,\mathbf{p}^{(k)})}$$

$$\mathbf{p}^{(k+1)} = \mathbf{z}^{(k+1)} + \beta^{(k)}\,\mathbf{p}^{(k)}$$

$$k = k + 1$$

Figure 4.3. The Preconditioned Conjugate Gradient Method

Theoretical considerations suggest that at the end of each of the first few iterations the residual norm is reduced by a factor of $\sqrt{[\kappa(\boldsymbol{K}\boldsymbol{B}^{-1}) - 1]} / \sqrt{[\kappa(\boldsymbol{K}\boldsymbol{B}^{-1}) + 1]}$. Note that when κ is unity a single iteration is sufficient to solve the equation. Of course κ is one only when $\boldsymbol{K} = \boldsymbol{B}$. However this provides us with a guideline for choosing $\boldsymbol{B}$. $\boldsymbol{B}$ must be chosen such that one can easily compute the solution of a linear system of equations with $\boldsymbol{B}$ as its matrix coefficient while at the same time it is as close to $\boldsymbol{K}$ as possible. These are contradictory requirements which makes the problem of finding good preconditioners a challenging one. For a well chosen $\boldsymbol{B}$ only a few iterations is required to reduce the residual norm to the desired level.

One of most simple preconditioners is the *diagonal* preconditioner. In this case the matrix $\boldsymbol{B}$ is a diagonal matrix which diagonal coefficients are those of the system matrix $\boldsymbol{K}$.

4.3. Multigrid Methods

Multigrid methods are fast iterative algorithms employed to obtain the solution of discrete boundary value problems. In this work we are concerned with the finite element

solution of structural mechanics problems, represented by linear matrix equation (4.1). This basic equation appears in the majority of engineering analyses, whether they be static, dynamic, linear or nonlinear.

Consider an approximate solution $\bar{x}$ to the exact solution x of equation (4.1), there are two important measures of $\bar{x}$ as an approximation to x. One is the *numerical error* and is given by

$$\Delta x = x - \bar{x} \tag{4.13}$$

Another computable measure of how well $\bar{x}$ approximates the exact solution x is the *residual*, given by

$$r = f - K\bar{x} \tag{4.14}$$

If we rearrange its definition as

$$K\bar{x} = f - r \tag{4.15}$$

and then subtract this expression from equation (4.1), we find an important relation between the numerical error and the residual

$$K\,\Delta x = r \tag{4.16}$$

which is called the *residual equation*. This equation plays a vital role in multigrid methods and it says that the numerical error Δx satisfies the same set of equations as the unknown x when f is replaced by the residual r.

Just as we can expand fairly arbitrary functions using a set of eigenfunctions, it is possible to expand arbitrary vectors in terms of the eigenvectors of the matrix K. In particular consider the expansion of the error vector, it can be proved [2] that many standard iterative methods, such as Jacobi or Gauss-Seidel methods are very effective at eliminating the high-frequency or oscillatory components of the error, while leaving the low-frequency or smooth components relatively unchanged.

One way to improve a relaxation scheme, at least in its early stages, is to use a good initial guess. A well-known technique for obtaining an improved initial guess is to perform some preliminary iterations on a coarse grid and then use the resulting approximation as an initial guess on the original fine grid. Relaxation on a coarser grid is less expensive since there are fewer unknowns to be updated.

Recalling that most basic relaxation schemes suffer in the presence of smooth components of the error, we now ask what these smooth components look like on a coarser grid. And the answer is that a smooth mode of the fine grid looks more oscillatory on the coarse grid. Noting that the k-th mode of both grids are approximations to the k-th mode of the continuos model, its evident that for the coarse grid this mode is more closer to the high-frequency end of its representable spectrum than for the fine grid. This suggests that when relaxation begins to stall on the fine mesh, signaling the predominance of smooth error modes, it is advisable to move to a coarser grid, on which those smooth error modes appear more oscillatory and relaxation will be more effective. Then we can improve the approximation $\bar{x}$ by solving the residual

equation on a coarser mesh. Denoting quantities on the coarse mesh by the subscript c we have

$$\boldsymbol{K}_c \, \Delta \boldsymbol{x}_c = \boldsymbol{r}_c \tag{4.17}$$

where $\boldsymbol{K}_c$ is the stiffness matrix of the coarse mesh and $\boldsymbol{r}_c$ is the residual of the fine mesh approximation transferred to the coarse grid by an operation called *restriction*. The restriction operation depends on the geometrical relations between the coarse and fine meshes, and the type of finite element employed. One way to define a coarse mesh is by eliminating alternate nodes from the fine mesh by applying constraints to the fine mesh degrees of freedom. Figure 4.4 shows a fine mesh of four triangles which are coarsened to form one triangle in the coarse mesh

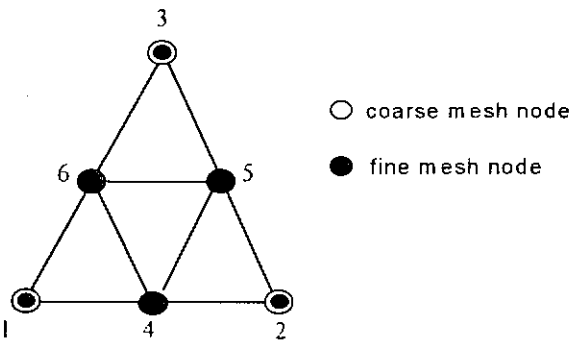


Figure 4.4. Coarsening for the linear triangle element.

In this case, the constraint equations take the form

$$x_f^1 = x_c^1 \tag{4.18.a}$$

$$x_f^2 = x_c^2 \tag{4.18.b}$$

$$x_f^3 = x_c^3 \tag{4.18.c}$$

$$x_f^4 = (x_c^1 + x_c^2)/2 \tag{4.18.d}$$

$$x_f^5 = (x_c^2 + x_c^3)/2 \tag{4.18.e}$$

$$x_f^6 = (x_c^3 + x_c^1)/2 \tag{4.18.f}$$

where subscripts f refers to the fine mesh. These equations defines the displacements of the coarse mesh at the locations of the nodes of the fine mesh. It must be noted that the constraints of Eqns. (4.18.d, e, f) assumes that nodes 4, 5, 6, are midside nodes of the parent triangle of the coarse mesh, if this is not the case the coefficients of these equations must be slightly different. In matrix form these relations can be expressed as

$$\Delta \boldsymbol{x}_f = \boldsymbol{T} \, \Delta \boldsymbol{x}_c \tag{4.19}$$

where $\boldsymbol{T}$ is the interpolation matrix .

To solve equation (4.17) we must know the residual forces on the coarse mesh transferred from the fine mesh. Since forces on the fine mesh must do the same work when they are restricted to the coarse mesh, it can be shown (*see* appendix *A*) that

$$\boldsymbol{r}_c = \boldsymbol{T}^{\mathrm{T}} \boldsymbol{r}_f \tag{4.20}$$

where $\boldsymbol{T}^{\mathrm{T}}$ is the *restriction* operator and is simply the transpose of the *interpolation* operator $\boldsymbol{T}$. The coarse mesh stiffness matrix $\boldsymbol{K}_c$ can then be computed as

$$\boldsymbol{K}_c = \boldsymbol{T}^{\mathrm{T}} \boldsymbol{K}_f \boldsymbol{T} \tag{4.21}$$

and solving equation (4.17) we obtain the coarse mesh correction $\Delta \boldsymbol{x}_c$. Then is possible to obtain an improved solution on the fine mesh as

$$\hat{\boldsymbol{x}}_f = \bar{\boldsymbol{x}}_f + \Delta \boldsymbol{x}_f \tag{4.22}$$

where $\Delta \boldsymbol{x}_f$ is the fine mesh correction obtained from the coarse grid correction $\Delta \boldsymbol{x}_c$ by the *interpolation* operation of equation (4.19).

The above ideas constitutes the basis for the multigrid method where cycles of relaxation on the fine and coarse meshes are combined with intergrid transfers to reduce the high and low frequency components of the error of an approximate solution.

4.3.1. Basic Multigrid Algorithm

To illustrate the method, consider a two mesh arrangement. Our objective is to solve equation (4.1) on a fine grid

$$\boldsymbol{K}_f \, \boldsymbol{x}_f = \boldsymbol{f}_f \tag{4.23}$$

The k-th cycle of the two level multigrid algorithm consists of the following steps:

- Starting with the current approximation $\boldsymbol{x}_f^{(k)}$ perform ν_1 cycles of an appropriate relaxation method on the fine mesh to obtain a new approximate solution $\bar{\boldsymbol{x}}_f^{(k)}$.
- Compute the residual $\bar{\boldsymbol{r}}_f^{(k)}$ associated with $\bar{\boldsymbol{x}}_f^{(k)}$ on the fine mesh

$$\bar{\boldsymbol{r}}_f^{(k)} = \boldsymbol{f} - \boldsymbol{K}_f \, \bar{\boldsymbol{x}}_f^{(k)} \tag{4.24}$$

- Restrict the fine mesh residual onto the coarse mesh using the fine-to-coarse mesh restriction operator $\boldsymbol{T}^{\mathrm{T}}$ to obtain the coarse mesh residual

$$\bar{\boldsymbol{r}}_c^{(k)} = \boldsymbol{T}^{\mathrm{T}} \, \bar{\boldsymbol{r}}_f^{(k)} \tag{4.25}$$

- Solve the coarse mesh residual equation to produce the coarse mesh correction

$$\boldsymbol{K}_c \Delta \bar{\boldsymbol{x}}_c^{(k)} = \bar{\boldsymbol{r}}_c^{(k)} \tag{4.26}$$

If the number of unknowns are sufficiently small we can use a direct solver in this step.

- Interpolate the coarse mesh correction to the fine mesh using the coarse-to-fine mesh interpolation operator $\boldsymbol{T}^{\mathrm{T}}$ to obtain the fine mesh correction

$$\Delta\bar{\boldsymbol{x}}_f^{(k)} = \boldsymbol{T}^{\mathrm{T}}\,\Delta\bar{\boldsymbol{x}}_c^{(k)} \tag{4.27}$$

- Compute the new approximate solution on the fine mesh

$$\hat{\boldsymbol{x}}_f^{(k)} = \bar{\boldsymbol{x}}_f^{(k)} + \Delta\bar{\boldsymbol{x}}_f^{(k)} \tag{4.28}$$

- Finally, perform ν_2 cycles of relaxation starting with $\hat{\boldsymbol{x}}_f^{(k)}$ to produce the new fine mesh solution $\boldsymbol{x}_f^{(k+1)}$.

This cycle is repeated m times until a convergence criterion is satisfied on the fine mesh, such as the residual criteria, that is

$$\frac{\left\|\bar{\boldsymbol{r}}_f^{(m)}\right\|}{\left\|\boldsymbol{f}_f\right\|} \le \varepsilon \tag{4.29}$$

where $\|\cdot\|$ denotes the Euclidean norm of a vector, and ε is an user supplied tolerance. In the present implementation we use another convergence criteria based in the energy norm of the increment that will be showed in section 4.5.

The basic two grid method can be extended to produce a true multigrid method by introducing still coarser meshes, and recursively using one or more cycles of the two grid method to obtain approximate solutions to the coarse mesh correction in equation (4.17). Figure 4.5 shows a graphical representation of the multigrid method.

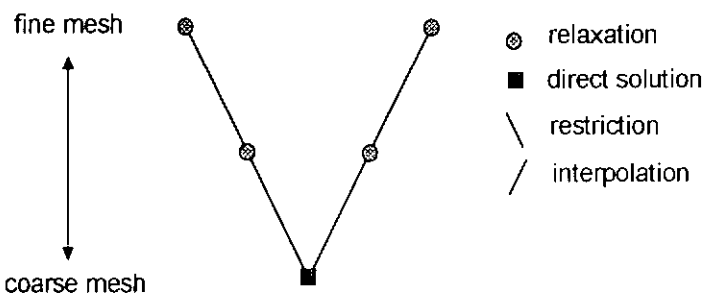


Figure 4.5. Graphical representation of the Multigrid Method

4.3.2 Adaptive Multigrid Methods

Adaptive multigrid methods combine the basic multigrid ideas of fine mesh relaxation and coarse mesh correction to solve equation (4.1) with an adaptive mesh refinement scheme to create the hierarchy of meshes required by the multigrid algorithm [62,75].

Since the elements are organized by levels of refinement is more effective to apply the multigrid transfers between *grids* formed by elements of the same level. In Fig. 4.6 shows a sequence of three adaptively generated meshes and their respective grids. Each grid m is formed for all the elements of level m and all the unrefined elements of levels lower than m.

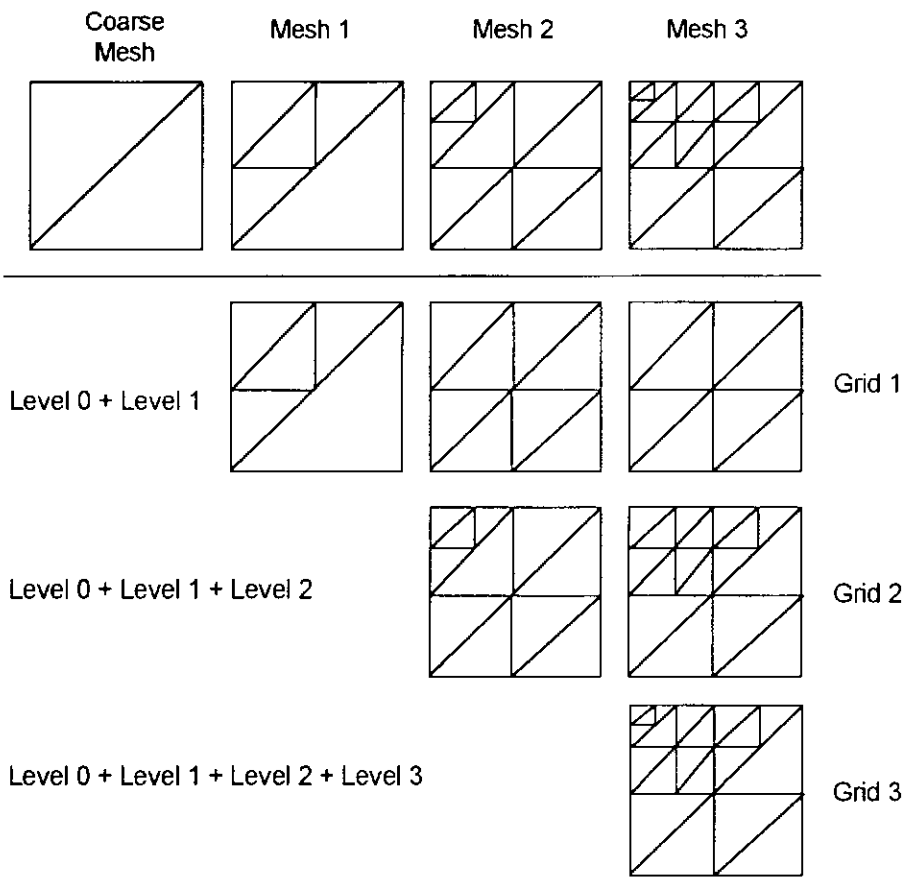


Figure 4.6. Sequence of meshes and grids used in the Adaptive Multigrid Method

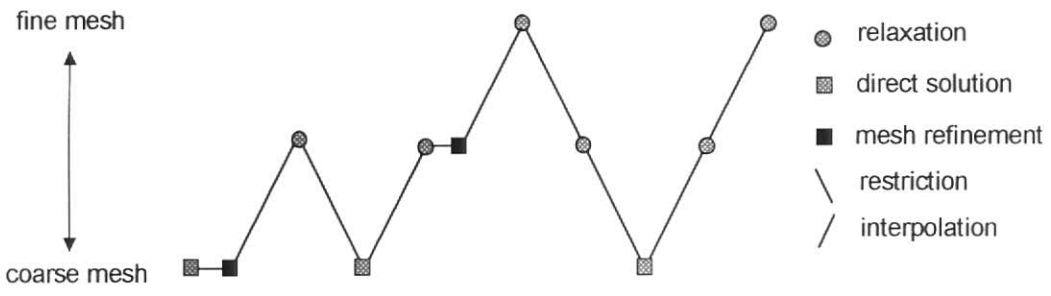


Figure 4.7. Graphical representation of the Adaptive Multigrid Method

In the adaptive multigrid algorithm, a mesh refinement is required after Eqn. (4.1) has been solved to produce a new fine mesh. Mesh refinement ceases only after the chosen error estimator indicates that a solution has been obtained within the user specified discretization error bounds.

4.4. Multi-level Preconditioners

The main objective of a preconditioner is to improve the rate of convergence of a basic relaxation algorithm. One of the most efficient preconditioners are the multi-level preconditioners, which give a proportional relation between the computational effort of the relaxations and the number of equations of the system to be solved [70]. To use this type of preconditioners is necessary to generate a sequence of refinements (uniform or adaptive) of an initial coarse mesh, and if instead of the standard shape functions used to generate the finite element matrices we use a hierarchical base then we can retain the information of the previous meshes as it will be shown in the next section.

4.4.1 Hierarchical Shape Functions

In its usual definition, finite element shape functions are associated with nodal values of the approximation. This identification has the merit of assigning a "physical" meaning to the amplitudes of the shape functions. There is, however, a disadvantage in this "standard" definition of the finite element shape functions. If we considered the succession of meshes generated by an uniform refinement, each level of approximation results in an entirely reevaluation of the equation set, that is

level 0
$$\boldsymbol{K}_{00}^{(0)}\,\boldsymbol{x}_0 = \boldsymbol{f}_0^{(0)} \tag{4.30.a}$$

level 1
$$\begin{bmatrix} \boldsymbol{K}_{00}^{(1)} & \boldsymbol{K}_{01}^{(1)} \\ \boldsymbol{K}_{10}^{(1)} & \boldsymbol{K}_{11}^{(1)} \end{bmatrix} \begin{Bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_1 \end{Bmatrix} = \begin{Bmatrix} \boldsymbol{f}_0^{(1)} \\ \boldsymbol{f}_1^{(1)} \end{Bmatrix} \tag{4.30.b}$$

level 2

$$\begin{bmatrix} \boldsymbol{K}_{00}^{(2)} & \boldsymbol{K}_{01}^{(2)} & \boldsymbol{K}_{02}^{(2)} \\ \boldsymbol{K}_{10}^{(2)} & \boldsymbol{K}_{11}^{(2)} & \boldsymbol{K}_{12}^{(2)} \\ \boldsymbol{K}_{20}^{(2)} & \boldsymbol{K}_{21}^{(2)} & \boldsymbol{K}_{22}^{(2)} \end{bmatrix} \begin{Bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{Bmatrix} = \begin{Bmatrix} \boldsymbol{f}_0^{(2)} \\ \boldsymbol{f}_1^{(2)} \\ \boldsymbol{f}_2^{(2)} \end{Bmatrix} \tag{4.30.c}$$

where the index between parenthesis indicates the level where each matrix or vector were formed.

But, it is possible to use "hierarchical" shape functions which amplitudes are not directly associated with nodal values. An important aspect of the hierarchical basis is that when a level is refined the new discretization preserves information of previous discretizations. The use of hierarchical basis results in the following sequence of approximations:

level 0

$$\hat{\boldsymbol{K}}_{00}^{(0)} \hat{\boldsymbol{x}}_0 = \hat{\boldsymbol{f}}_0^{(0)} \tag{4.31.a}$$

level 1

$$\left[\begin{array}{c|c} \hat{\boldsymbol{K}}_{00}^{(0)} & \hat{\boldsymbol{K}}_{01}^{(1)} \\ \hline \hat{\boldsymbol{K}}_{10}^{(1)} & \hat{\boldsymbol{K}}_{11}^{(1)} \end{array}\right] \begin{Bmatrix} \hat{\boldsymbol{x}}_0 \\ \hat{\boldsymbol{x}}_1 \end{Bmatrix} = \begin{Bmatrix} \hat{\boldsymbol{f}}_0^{(0)} \\ \hat{\boldsymbol{f}}_1^{(1)} \end{Bmatrix} \tag{4.31.b}$$

level 2

$$\left[\begin{array}{cc|c} \hat{\boldsymbol{K}}_{00}^{(0)} & \hat{\boldsymbol{K}}_{01}^{(1)} & \hat{\boldsymbol{K}}_{02}^{(2)} \\ \hat{\boldsymbol{K}}_{10}^{(1)} & \hat{\boldsymbol{K}}_{11}^{(1)} & \hat{\boldsymbol{K}}_{12}^{(2)} \\ \hline \hat{\boldsymbol{K}}_{20}^{(2)} & \hat{\boldsymbol{K}}_{21}^{(2)} & \hat{\boldsymbol{K}}_{22}^{(2)} \end{array}\right] \begin{Bmatrix} \hat{\boldsymbol{x}}_0 \\ \hat{\boldsymbol{x}}_1 \\ \hat{\boldsymbol{x}}_2 \end{Bmatrix} = \begin{Bmatrix} \hat{\boldsymbol{f}}_0^{(0)} \\ \hat{\boldsymbol{f}}_1^{(1)} \\ \hat{\boldsymbol{f}}_2^{(2)} \end{Bmatrix} \tag{4.31.c}$$

In each level above it can be seen that, as the approximation is refined, the matrices and vectors generated in the previous levels reoccur and need not be recomputed.

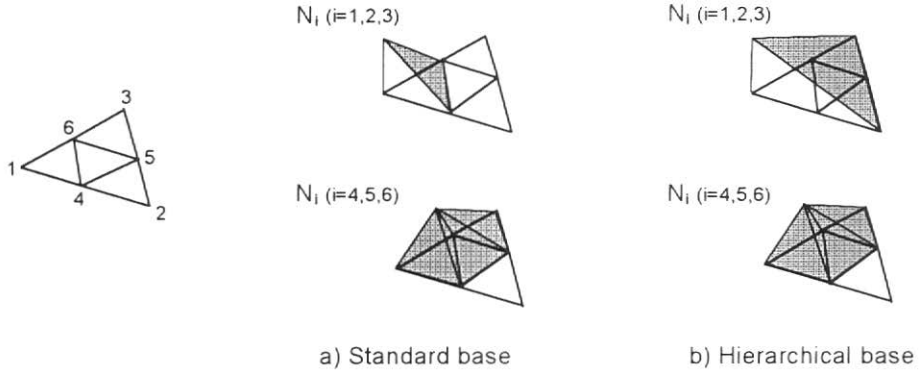


Figure 4.8 . Standard and hierarchical shape functions on a subdivided triangle.

Figure 4.8 shows a standard nodal base and a hierarchical base for a regular subdivided triangular element. In Figure 4.9 we can see the variation along a side of a function ϕ approximated hierarchically over a subdivided element.

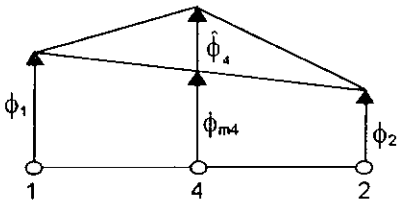


Figure 4.9 - Variation along a side of a linear hierarchical approximation.

The value ϕ_4 of the function ϕ at the midside node is composed of two values: the hierarchical part $\hat{\phi}_4$ and the linear approximation ϕ_{m4} at the midside node, which is the average of the total values at the end nodes ϕ_1 and ϕ_2, that is

$$\phi_{m4} = (\phi_1 + \phi_2) / 2 \tag{4.32.a}$$

$$\phi_4 = \hat{\phi}_4 + \phi_{m4} \tag{4.32.b}$$

These relations between variables of the standard and hierarchical basis can be expressed in matrix form as:

$$\boldsymbol{x} = \boldsymbol{S}\,\hat{\boldsymbol{x}} \tag{4.33}$$

The non-singular transformation matrix $\boldsymbol{S}$ can be partitioned as:

$$\boldsymbol{S} = \begin{bmatrix} \boldsymbol{I}_0 & \boldsymbol{0} \\ \boldsymbol{W} & \boldsymbol{I}_1 \end{bmatrix} \tag{4.34}$$

where $\boldsymbol{I}_0$ and $\boldsymbol{I}_1$ are identity matrices which correspond to the initial and superior levels, respectively, and the submatrix $\boldsymbol{W}$ contains weights which represents relations of the type of eqn.(4.32.a) for each degree of freedom of the finer meshes, that is

$$\boldsymbol{S} = \boldsymbol{S}_n\,\boldsymbol{S}_{n-1}\,\boldsymbol{S}_{n-2}\,\ldots\ldots\,\boldsymbol{S}_1 \tag{4.35}$$

Where the indices 1 to n represent the degrees of freedom of the variables of the finer meshes, ordered sequentially from the first refined mesh to the last. Then to obtain the nodal base representation of a displacement vector expressed in hierarchical coordinates we must apply successive transformations $\boldsymbol{S}_i$ from the first refined level to the last

$$\boldsymbol{x} = \boldsymbol{S}\,\hat{\boldsymbol{x}} = \boldsymbol{S}_n\,\boldsymbol{S}_{n-1}\,\boldsymbol{S}_{n-2}\,\ldots\ldots\,\boldsymbol{S}_1\,\hat{\boldsymbol{x}} \tag{4.36}$$

The transpose matrix $\boldsymbol{S}^T$ is

$$\boldsymbol{S}^T = \boldsymbol{S}_1^T\boldsymbol{S}_2^T\boldsymbol{S}_3^T\ldots\ldots\,\boldsymbol{S}_n^T \tag{4.37}$$

then to obtain the hierarchical representation $\hat{f}$ of a force vector f expressed in terms of nodal standard variables we must apply successive transformations S_i^T from the last refined level to the first

$$\hat{f} = S^T f = S_1^T S_2^T S_3^T \ldots\ldots S_n^T f \tag{4.38}$$

Then, to obtain the solution of eqn.(4.1) we pre-multiply this equation by S^T obtaining

$$S^T K x = S^T K S \hat{x} = S^T f \tag{4.39}$$

which can be expressed as

$$\hat{K}\hat{x} = \hat{f} \tag{4.40}$$

Where the matrix $\hat{K}$ has the form showed by the system matrix of eqns.(4.31), that is

$$\hat{K} = \begin{bmatrix} \hat{K}_{00}^{(0)} & \hat{K}_{01}^{(1)} & \hat{K}_{02}^{(2)} \\ \hat{K}_{10}^{(1)} & \hat{K}_{11}^{(1)} & \hat{K}_{12}^{(2)} \\ \hat{K}_{20}^{(2)} & \hat{K}_{21}^{(2)} & \hat{K}_{22}^{(2)} \end{bmatrix} \tag{4.41}$$

and due to the properties of the hierarchical basis $\hat{K}$ is in general a much more dense matrix than the original matrix K. It must be noted that since the variables of the lower level are not altered we have

$$\hat{K}_{00}^{(0)} = K_{00}^{(0)} \tag{4.42}$$

An important advantage of the hierarchical basis is that the matrices obtained at each stage of refinement are better conditioned than that obtained with the standard nodal basis [67].

Also note, that for constrained variables on transition sides, its associated hierarchical variables are all null, since the constrained varible assumes an averaged value from the side extreme variables.

4.4.2 A Hierarchical Multi-level Preconditioner

We can define a hierarchical preconditioner matrix $\hat{B}$ as

$$\hat{B} = \begin{bmatrix} (LDL^T)_{00}^{(0)} & 0 & 0 \\ 0 & \hat{D}_{11}^{(1)} & 0 \\ 0 & 0 & \hat{D}_{22}^{(2)} \end{bmatrix} \tag{4.43}$$

where $(LDL^T)_{00}^{(0)}$ represents the Crout factorization of the matrix $K_{00}^{(0)}$, and $\hat{D}_{11}^{(1)}$, $\hat{D}_{22}^{(2)}$ are the diagonal elements of the matrices $\hat{K}_{11}^{(1)}$, $\hat{K}_{22}^{(2)}$, respectively.

Then the solution of the auxiliar system $Bz = r$ of the preconditioned iterative methods can be made following the next steps:

(1). Transformation of vector $\boldsymbol{r}$ from nodal to hierarchical base

$$\hat{\boldsymbol{r}} = \boldsymbol{S}^{\mathrm{T}} \boldsymbol{r} = \boldsymbol{S}^{\mathrm{T}} \begin{Bmatrix} \boldsymbol{r}_0 \\ \boldsymbol{r}_1 \end{Bmatrix} \tag{4.44}$$

(2). Solve the coarse mesh system

$$\boldsymbol{K}_{00}^{(0)} \hat{\boldsymbol{z}}_0 = (\boldsymbol{L}\boldsymbol{D}\boldsymbol{L}^{\mathrm{T}})_{00}^{(0)} \hat{\boldsymbol{z}}_0 = \hat{\boldsymbol{r}}_0 \equiv \boldsymbol{r}_0 \tag{4.45}$$

(3). Solve the diagonal system of the finer meshes

$$\hat{\boldsymbol{z}}_1 = (\hat{\boldsymbol{D}}_{11}^{(1)})^{-1} \hat{\boldsymbol{r}}_1 \tag{4.46.a}$$

$$\hat{\boldsymbol{z}}_2 = (\hat{\boldsymbol{D}}_{22}^{(2)})^{-1} \hat{\boldsymbol{r}}_2 \tag{4.46.b}$$

(4). Transformation of vector $\boldsymbol{z}$ from hierarchical to nodal base

$$\boldsymbol{z} = \boldsymbol{S}\, \hat{\boldsymbol{z}} = \boldsymbol{S} \begin{Bmatrix} \hat{\boldsymbol{z}}_0 \\ \hat{\boldsymbol{z}}_1 \end{Bmatrix} \tag{4.47}$$

It must be noted that with this strategy there is no need of compute the matrices in the hierarchical basis, only vector transformations are employed.

4.5. Convergence criteria

A very common procedure used to terminate an iterative method for solving Eqn.(4.1) is to compute the residual $\boldsymbol{r}^{(k)}$ associated with the approximate solution $\boldsymbol{x}^{(k)}$ after k cycles and to stop iterating when

$$\|\boldsymbol{r}^{(k)}\| / \|\boldsymbol{f}\| < \varepsilon \tag{4.48}$$

where $\| \cdot \|$ is the Euclidean norm and ε is an user supplied tolerance. Using this approach, solution is often possible to within machine accuracy.

For engineering purposes the objective of a numerical analysis is to obtain a discretized solution of partial differential equations within a discretization error. Usually the discretization error is measured in the energy norm $\| \cdot \|_{\mathrm{K}}$ defined for an arbitrary error vector $\boldsymbol{e}$ as

$$\| \boldsymbol{e} \|_{\mathrm{K}} = (\boldsymbol{e}^{\mathrm{t}} \boldsymbol{K}\, \boldsymbol{e})^{1/2} \tag{4.49}$$

Note that this definition is equivalent to the definition given in Eqn.(2.4) for the energy norm of the error.

The error vector $\boldsymbol{e}$ represents the difference between the discrete solution obtained from the discretized problem and the exact solution of the continuous problem. Since the exact solution of the continuous problem is generally not available, several estimates are used in practice giving an estimated error $\boldsymbol{e}^*$. These estimates are based on the solution of the system of equations (4.1), and generally an approximate but precise solution of these equations suffices to provide an adequate estimation of the discretization error.

Then an approximated solution $x^{(k+1)}$ can be considered acceptable if satisfies (see Eqn. 2.19)

$$\| e^* \|_K < \eta \| x \|_K \approx \eta \| x^{(k+1)} \|_K \tag{4.50}$$

where η is the discretization error tolerance in the energy norm.

In the gradient methods we are looking for a minimum of the total potential energy ϕ, Eqn.(4.2), then makes sense to monitor the energy variation along the iterative process and terminate the iterations when the solution have reached a stabilized value in the energy norm. That is, if $x^{(k)}$ and $x^{(k+1)}$ are two successive solutions of the iterative process we monitor if the next inequality is satisfied in each step

$$\| x^{(k+1)} - x^{(k)} \|_K < \varepsilon_U \| x^{(k+1)} \|_K \tag{4.51}$$

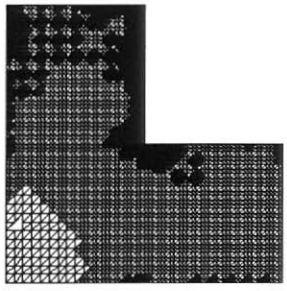
where ε_U is the numerical tolerance in the energy norm.

Strictly speaking, a small value of the left hand side of the above inequality, which represents the energy variation between two successive steps, does not means that the right hand side has reached a stabilized value, since the variation of the left hand side is, in general, not monotonic with the iterations. But, as it will be shown in some numerical examples, for practical purposes this variation can be considered almost monotonic. Also exists numerical evidence [76] that the relative energy norm of the increment is approximately equal to the relative error in the energy norm of the numerical solution, that is

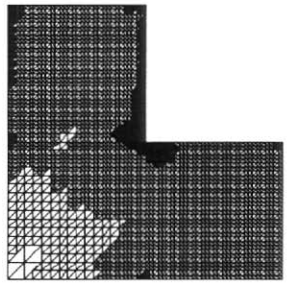
$$\| x^{(k+1)} - x_h \|_K / \| x_h \|_K \approx \| x^{(k+1)} - x^{(k)} \|_K / \| x^{(k+1)} \|_K < \varepsilon_U \tag{4.52}$$

where x_h is the exact numerical solution for this mesh.

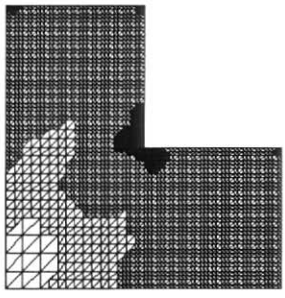
Then a tentative value for the numerical tolerance ε_U could be $\varepsilon_U = \eta$, but since the precision of the approximate solution $x^{(k+1)}$ affects the precision of the error estimator, in general a much smaller value must be imposed to ε_U to obtain acceptable results. To analize this influence we considered the L-shaped domain problem of chapter 2. Fig. 4.10 shows the final meshes obtained with different values of the numerical tolerance ε_U to achieve an estimated discretization error of 5%. It can be seen that the final mesh converges to a unique discretization, and for rather different values of ε_U almost the same meshes are obtained, this can leads to the erroneous conclusion that the optimal value of ε_U would be the higher that not affects significantly the discretization, since for a high value of ε_U less iterations are needed. Figs. 4.11 and 4.12 show the influence of the numerical tolerance on the behaviour of the adaptive refinement procedure and the performance of the conjugate gradients method with hierarchical preconditioning. From the analysis of these figures we could conclude that an optimal value of ε_U =0.01 (1 %) must be adopted, since it requires less iterations without affecting significantly the discretization. It must be noted that a similar criteria was adopted for Papadrakakis and Babilis [77] to determine the numerical tolerance when applying iterative solvers with *p*-adaptivity.



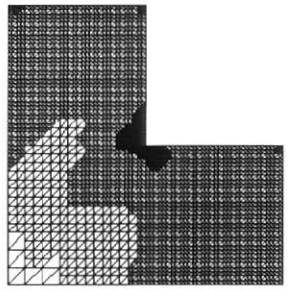
ε_U = 5 % η = 24.69 %
DOF = 11148



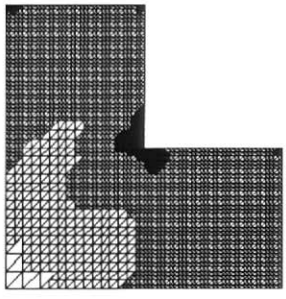
ε_U = 3 % η = 18.95 %
DOF = 7454



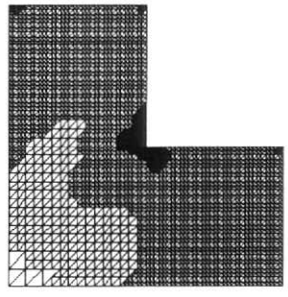
ε_U = 1 % η = 9.80 %
DOF = 6320



ε_U = 0.5 % η = 7.87 %
DOF = 6234



ε_U = 0.1 % η = 4.81 %
DOF = 6222



ε_U = 0.01 % η = 5.00 %
DOF = 6224

Figure 4.10. L-shaped domain. Side flux averaging error estimator.

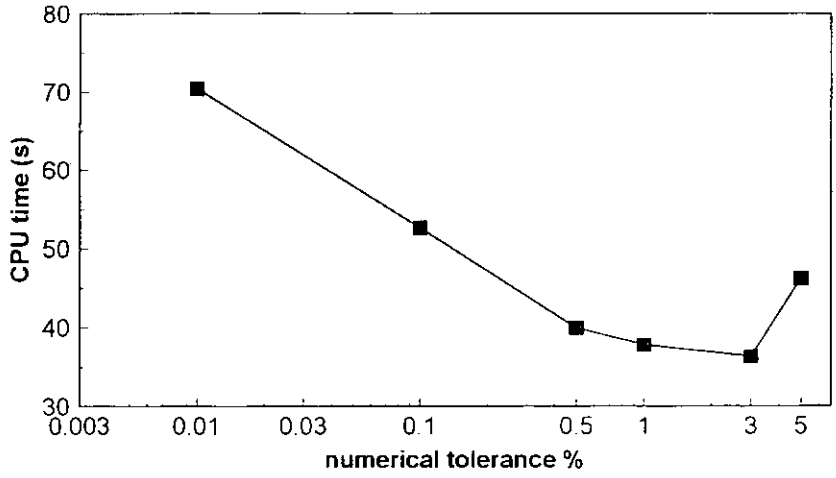


Figure 4.11. Variation of degrees of CPU time with numerical tolerance. Side flux averaging error estimator.

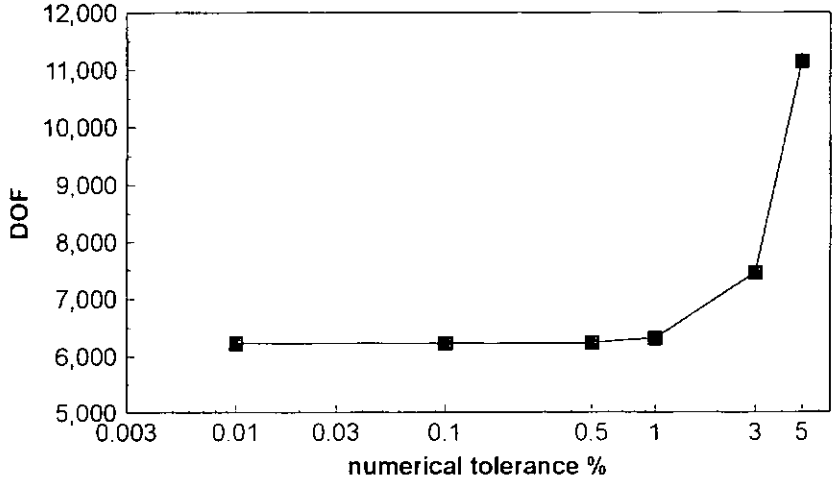


Figure 4.12. Variation of degrees of freedom in fine mesh with numerical tolerance. Side flux averaging error estimator.

But, as it is shown in Fig. 4.10, the exact discretization error is far from the expected bound of 5% for high values of ε_U, indicating a great deterioration of the error estimator and of the numerical solution, even for similar meshes.

In Figs. 4.13 and 4.14 the histories of the effectivity indices for the side flux averaging and the superconvergent patch recovery error estimators are shown. The values in the graph corresponds to the numerical tolerance of each curve. It can be seen that a severe lost of asymptotic exactness ocurrs for high values of the numerical tolerance, and that a minimum value of ε_U =0.001 (0.1 %) must be adopted to retain the accuracy of the error estimator along the refinement process.

It must be noted, that in an optimal strategy the numerical tolerance must be variable since for coarse meshes the precision of the estimator it is not so severely affected by the numerical tolerance. This has been implemented by Silva et al. [78] using the relative Euclidean norm of the residual as the controlling quantity and adopting a linear variation of the numerical tolerance with the estimated discretization error.

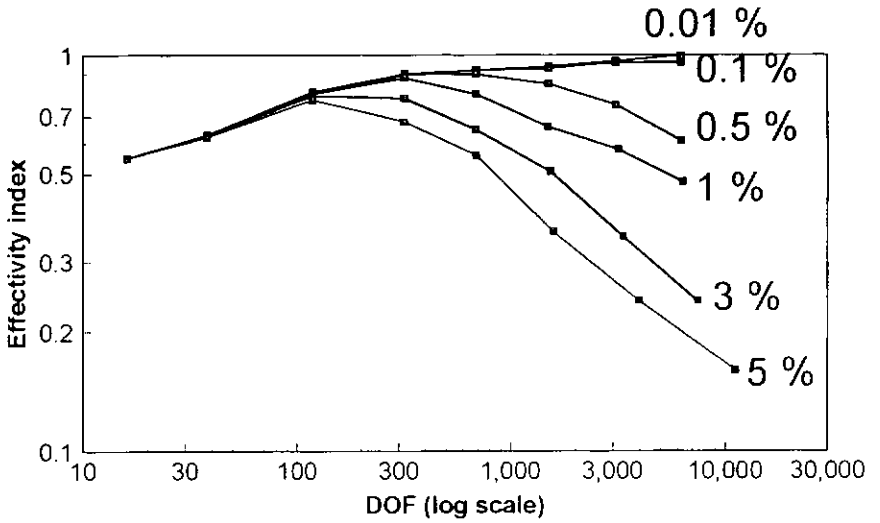


Figure 4.13. Variation of effectivity index with numerical tolerance. Side flux averaging error estimator.

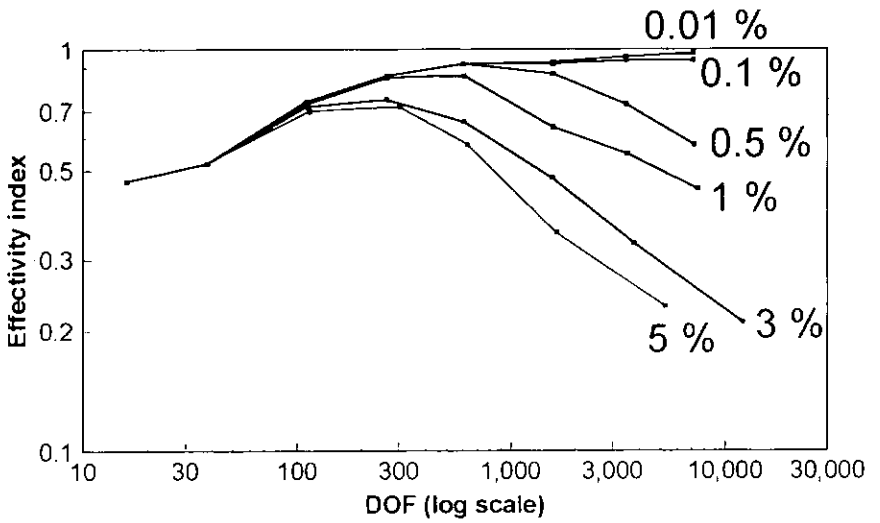


Figure 4.14. Variation of effectivity index with numerical tolerance. Superconvergent patch recovery error estimator.

4.5.1. Implementation of the convergence criteria

If we define U_{cur} as the energy of the current computed solution in step k, and U_{err} as the energy of the difference between two successive steps, that is

$$U_{cur} = (\boldsymbol{x}^{(k+1)})^t \boldsymbol{K} \boldsymbol{x}^{(k+1)} \tag{4.53.a}$$

$$U_{err} = (\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)})^t \boldsymbol{K} (\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}) \tag{4.53.b}$$

then the inequality (4.51) can be expressed as

$$U_{err} < \varepsilon_U^2 \, U_{cur} \tag{4.54}$$

It must be noted that the quantities involved in the above inequality are inexpensively obtained during each iterative step of the preconditioned conjugate gradient method, requiring only an additional dot product per step. If we define $\boldsymbol{f}_i^{(k+1)}$ as the vector of internal forces of the current solution $\boldsymbol{x}^{(k+1)}$ at step k, that is

$$\boldsymbol{f}_i^{(k+1)} = \boldsymbol{K} \boldsymbol{x}^{(k+1)} \tag{4.55}$$

and noting that the current solution $\boldsymbol{x}^{(k+1)}$ can be expressed as

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \alpha^{(k)} \boldsymbol{p}^{(k)} \tag{4.56}$$

then the energies U_{cur} and U_{err} can be written as

$$U_{cur} = (\boldsymbol{x}^{(k+1)})^t \boldsymbol{f}_i^{(k+1)} \tag{4.57}$$

$$U_{err} = (\alpha^{(k)})^2 (\boldsymbol{p}^{(k)})^t \boldsymbol{K} \boldsymbol{p}^{(k)} \tag{4.58}$$

where the vector $\boldsymbol{p}^{(k)}$ is the conjugated direction obtained in step k of the conjugate gradient method. The modified algorithm is shown in figure 4.15.

Given: $\mathbf{x}^{(0)}$, $k = 0$

Initialize:

$$\mathbf{f}_i^{(0)} = \mathbf{K}\,\mathbf{x}^{(0)}$$

$$\mathbf{r}^{(0)} = \mathbf{f} - \mathbf{f}_i^{(0)}$$

$$\mathbf{z}^{(0)} = \mathbf{B}^{-1}\,\mathbf{r}^{(0)}$$

$$\mathbf{p}^{(0)} = \mathbf{z}^{(0)}$$

$$aux2 = (\mathbf{r}^{(0)}, \mathbf{z}^{(0)})$$

Iterate over:

$$\mathbf{v}^{(k)} = \mathbf{K}\,\mathbf{p}^{(k)}$$

$$aux1 = (\mathbf{p}^{(k)}, \mathbf{v}^{(k)})$$

$$\alpha^{(k)} = \frac{aux2}{aux1}$$

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)}\,\mathbf{p}^{(k)}$$

$$\mathbf{f}_i^{(k+1)} = \mathbf{f}_i^{(k)} + \alpha^{(k)}\,\mathbf{v}^{(k)}$$

convergence ckeck:

$$U_{cur} = (\mathbf{x}^{(k+1)}, \mathbf{f}_i^{(k+1)})$$

$$U_{err} = (\alpha^{(k)})^2\, aux1$$

if $(U_{err} < \varepsilon_U^2\, U_{cur})$ stop

$$\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha^{(k)}\,\mathbf{v}^{(k)}$$

$$\mathbf{z}^{(k+1)} = \mathbf{B}^{-1}\,\mathbf{r}^{(k+1)}$$

$$aux3 = (\mathbf{r}^{(k+1)}, \mathbf{z}^{(k+1)})$$

$$\beta^{(k)} = \frac{aux3}{aux2}$$

$$\mathbf{p}^{(k+1)} = \mathbf{z}^{(k+1)} + \beta^{(k)}\,\mathbf{p}^{(k)}$$

$$aux2 = aux3$$

$$k = k + 1$$

Figure 4.15. Preconditioned Conjugate Gradient Method with energy monitoring.

4.6. Multilevel Newton-Raphson Method for Non-linear Problems

In this method the multilevel solver is used as an equation solver embedded inside Newton iterations. Proportional loading is considered and the size of the load steps are controlled by a loading parameter λ. For Each increment k of loading, the linear equation

$$\boldsymbol{K}_T^{(k)} \ \Delta \boldsymbol{x}_k^{(m+1)} = \lambda_k \boldsymbol{f}_0 \ - \boldsymbol{f}_i^{(m)} \tag{4.59}$$

is solved for $\Delta \boldsymbol{x}_k^{(m+1)}$ using the multilever solver, and a new set of displacements

$$\boldsymbol{x}_k^{(m+1)} = \boldsymbol{x}_k^{(m)} + \Delta \boldsymbol{x}_k^{(m+1)} \tag{4.60}$$

are computed. This procedure is repeated until the internal forces are in equilibrium with the externally applied loads to within some specified tolerance. The subscript k indicates the load step, while the superscript m denotes the Newton-Raphson iteration number. The parameter λ_k represents the applied load level and $\boldsymbol{f}_0$ is the loading pattern. The matrix $\boldsymbol{K}_T^{(k)}$ is the tangent stiffness matrix which is computed at the beginning of the step for the *modified* Newton-Raphson method or is updated in each Newton iteration for the *full* Newton-Raphson method. The vector $\Delta \boldsymbol{x}_k^{(m+1)}$ is the displacement increment, $\boldsymbol{x}_k^{(m)}$ and $\boldsymbol{x}_k^{(m+1)}$ are the displacements before and after the current Newton iteration, and $\boldsymbol{f}_i^{(m)}$ are the internal forces associated with the displacements $\Delta \boldsymbol{x}_k^{(m)}$.

The accuracy of the multilevel Newton-Raphson method depend upon various convergence criteria. Two iteration loops can be identified: one outer loop of Newton-Raphson iterations and an inner loop of iterations of the multilever solver. The convergence test applied to the outer loop determines the accuracy of the solution to the nonlinear problem, and examines the ratio of the Euclidean norm of incremental displacements to that of the accumulated increments, i.e., Newton-Raphson iteration is terminated when

$$\Delta \boldsymbol{x}_k^{(m)} < \varepsilon_0 \sum_{i=1}^{m} \Delta \boldsymbol{x}_k^{(i)} \tag{4.61}$$

where ε_0 is the outer loop tolerance specified by the user. The multilevel iterations are terminated with the energy criteria discussed in the previous sections.

It must be noted that when combined with adaptivity we must verify if the mesh at the beginnig of the Newton-Raphson step satisfies the discretization error tolerance at the end of the Newton iterations. If the discretization error is not satisfied we refine the mesh in this last configuration and the Newton-Raphson iterations are repeated from the beginning of the step with this new mesh.

Chapter 5

Numerical Results

5.1 Introduction

We divide this chapter in two parts, in the first part we make some comparisons of the performance of the multilevel solvers in linear problems with mesh adaptations and, in the second part we show some examples of geometrically non-linear problems.

5.2. Comparisons of Multilevel Solvers for Linear Adaptive Analysis

We use the following convention for the different solvers employed

PCG-H : conjugate gradients with hierarchical preconditioner.
PCG-D : conjugate gradients with diagonal preconditioner.
MG-H : multigrid with hierarchical PCG for relaxation.
MG-D : multigrid with diagonal PCG for relaxation.

In both multigrid solvers we perform five iterations in each relaxation step, that is, we made $\nu_1 = 5$, $\nu_2 = 5$, and the convergence check is made in each iteration on the finer mesh. In all solvers we use the energy criteria for convergence with a numerical tolerance $\varepsilon_U = 0.001$ (0.1% in energy norm).

It must be noted that the stiffness matrix $\boldsymbol{K}$ is only used as a linear operator for vector-matrix products. Noting that $\boldsymbol{K}$ is an assembly of element matrices the product $\boldsymbol{Kx}$ can be written as:

$$\boldsymbol{Kx} = \sum_e K^e x^e \tag{5.1}$$

where the superscript e denotes element quantities, and summation is carried over all the elements of the mesh. There are basically two approaches for the computation of the summation. One is the *element-by-element* method where the individual element matrices are calculated once and stored. The product $\boldsymbol{K^e x^e}$ is then directly computed. Another approach is the *matrix-free* method, in this case the individual element matrices are not explicitly formed and stored, instead the product $\boldsymbol{K^e x^e}$ is computed every time that is needed.

In this implementation the matrix-free approach was adopted due to the limited memory of the platform employed in the computations, an IBM®-PC compatible (DX4-100 mhz, 8 Mb Ram). It must be noted that numerical experiments of Parsons [81] considering an eight brick node isoparametric element indicates that the matrix-free computations require not only minimum storage but also few operations than the element-by-element method.

5.2.1 L - shaped region in plane stress

We consider the standard L-shaped domain problem which was also tested in chapter 2. The geometry, loads, boundary conditions and the initial mesh are shown in Fig. 5.1.

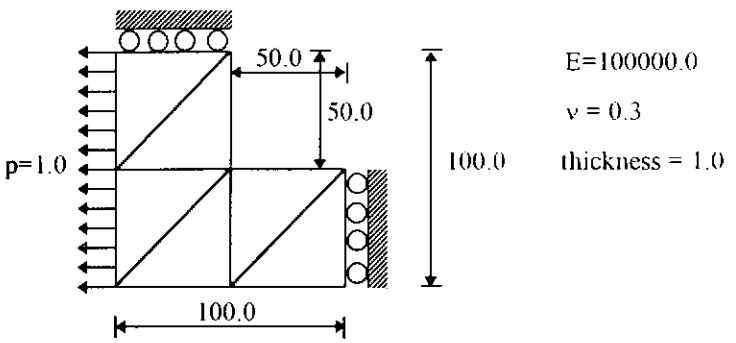


Figure 5.1 - L-shape domain problem.

In this example the initial mesh was adaptively refined using a discretization tolerance η = 5%, and the midside flux averaging error estimator. This gives seven levels of refinement which are shown in Fig. 2.25. of chapter 2.

In Figure 5.2 the total CPU time spent for each solver is shown. We can see that the diagonal PCG was the slowest of all the algorithms, and the hierarchical PCG was the fastest. Both multigrid algorithms were slower than the hierarchical PCG, and in particular for this example the MG-D solver spent almost 34% more time that the PCG-H solver and the MG-H solver spent almost 23% more time than the PCG-H solver. In Table 5.1 the total number of iterations employed by each solver on the fine mesh of each intermediate level are shown, where the number between parenthesis in the columns of the multigrid solvers are the number of multigrid V-cycles employed.

We can observe that the number of iterations on each mesh remains almost constant for the PCG-H solver, and the same applies for the number of V-cycles for the MG-H solvers. This can be attributed to the proper combination of hierarchical preconditioners, adaptive refinement criteria and numerical tolerance for iterations. The increase in time for the MG-H solver over the PCG-H solver can be attributed to the fact that preconditioners are more effective for PCG solvers than for multigrid solvers, since the smoothing properties of the intermediate multigrid relaxations reduce the effectivity of

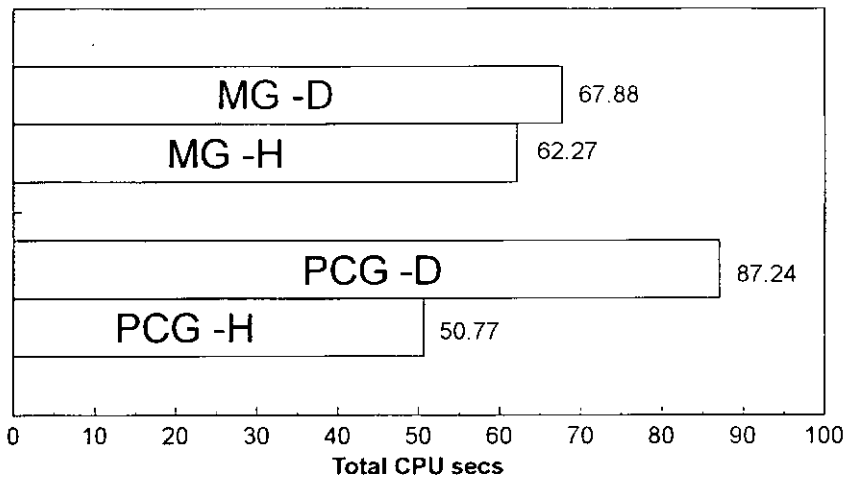


Figure 5.2 - L-shape problem. Total CPU time for linear adaptive analysis.

Table 5.1 - L-shape problem - Number of iterations on fine mesh of each level.

	ITERATIONS ON FINE MESH OF EACH LEVEL			
MESH (DOF)	PCG - H	PCG - D	MG - H	MG - D
1 (38)	12	22	15 (2)	31 (4)
2 (118)	19	44	20 (2)	39 (4)
3 (318)	21	90	18 (2)	31 (4)
4 (618)	21	119	17 (2)	18 (2)
5 (1502)	20	129	11 (2)	11 (2)
6 (3086)	19	35	11 (2)	9 (1)
7 (6222)	16	29	8 (1)	8 (1)
TOTAL TIME	50.77 secs.	87.24 secs.	62.27 secs.	67.88 secs.

Table 5.2 - L-shape problem - Memory storage for fine mesh of each level.

	MEMORY STORAGE (Bytes)					
MESH (DOF)	PCG		MG - H		MG - D	
1 (38)	7252	(1.00)	7812	(1.08)	8244	(1.14)
2 (118)	22458	(1.00)	24330	(1.08)	25738	(1.15)
3 (318)	59400	(1.00)	64920	(1.09)	68952	(1.16)
4 (618)	127648	(1.00)	142896	(1.12)	153264	(1.20)
5 (1502)	278026	(1.00)	320954	(1.15)	348426	(1.25)
6 (3086)	569448	(1.00)	674872	(1.19)	739928	(1.30)
7 (6222)	1145818	(1.00)	1409706	(1.23)	1566538	(1.37)

more elaborated preconditioners. This can be observed comparing the times of the MG-H and MG-D solvers, where we can see that the gain in effectivity of the hierarchical preconditioner is no so pronounced as in the case of the PCG solvers.

In Table 5.2 the memory requirements of each solver are shown. It must be noted that both PCG solvers require the same amount of storage since a direct solver is used for the initial coarse mesh to initiate the adaptive process. The numbers in parenthesis in the columns of the multigrid solvers indicate the proportional increment of memory with respect to the PCG solver for the same level. It must be noted that the requirement of storage of the multigrid solvers increases exponentially with the degrees of freedom, this is due to the fact that for each individual grid we need a displacement and a force vector to make the intergrid transfers, where for the PCG solvers only one displacement and one force vector for the entire mesh is needed. Also the MG-D solver needs a vector of diagonal scaling for each individual grid, and this results in a lack of proportionality of the storage requirements with the degrees of freedom since a same node can be repeated in various grids. In Fig. 5.3 we can see the variation of memory storage requirements of each solver during the adaptive process.

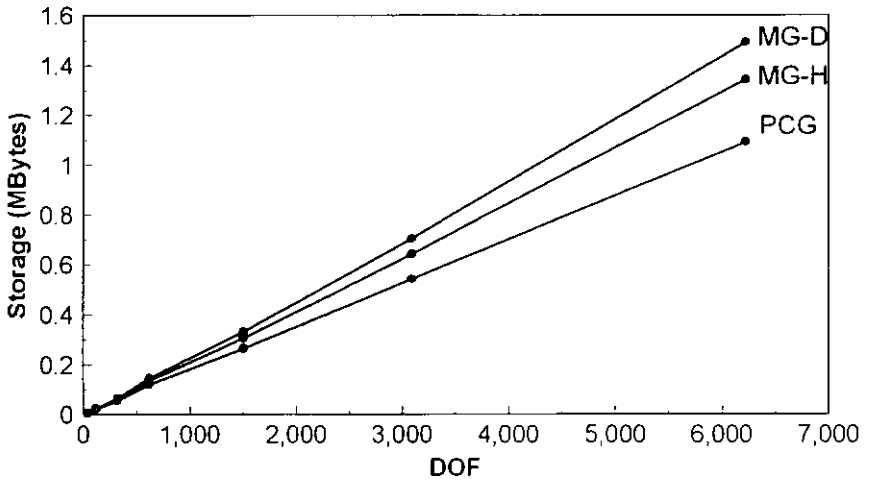


Figure 5.3 - L-shape problem - Memory storage.

For this example we also made a comparison of the convergence rate of the PCG solver in the final mesh. We take the relative energy norm of the increment $\| x^{(k+1)} - x^{(k)} \|_K / \| x^{(k+1)} \|_K$ which was the measure adopted to signalize convergence of the solution, and the error in the energy norm of the solution $\| x^{(k)} - x_h \|_K / \| x_h \|_K$. To compute the exact numerical value x_h we iterate till obtain a stabilized solution in the energy norm. For this example the value computed for the energy norm of the solution on the finer mesh was $\| x_h \|_K = 0.5572743533418$.

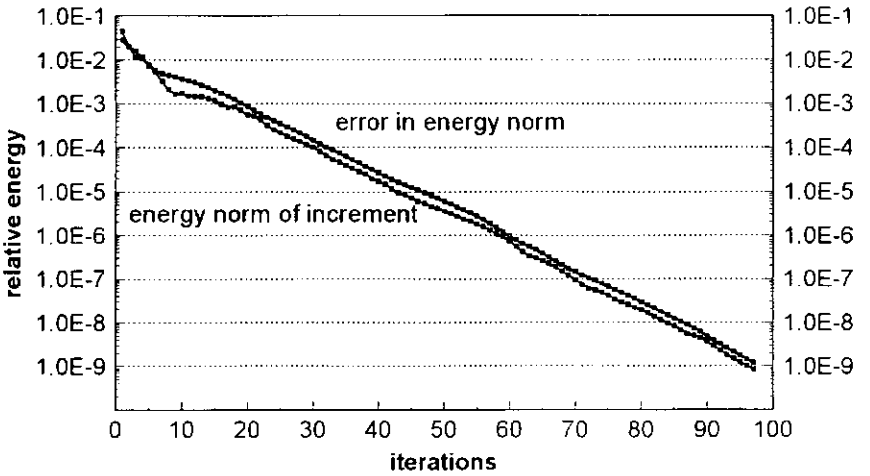


Figure 5.4 - Comparison between relative error in energy norm and relative energy norm of increment.

We can see in Fig.5.4 that both measures are almost identical and the variation of the relative energy norm of the increment can be assumed monotonic with the number of iterations. The comparisons were made using the hierarchical PCG. The same conclusion was arrived in [76] where uniform refinement was used to create the hierarchy of meshes for the multilevel solvers.

5.2.2. Square plate with a circular hole in plane stress.

We consider a plate with a circular hole subject to uniform tractions in all boundary sides. Due to symmetry conditions only one quarter of the plate was analyzed. Geometry, essential and natural boundary conditions are given in Fig. 5.5.

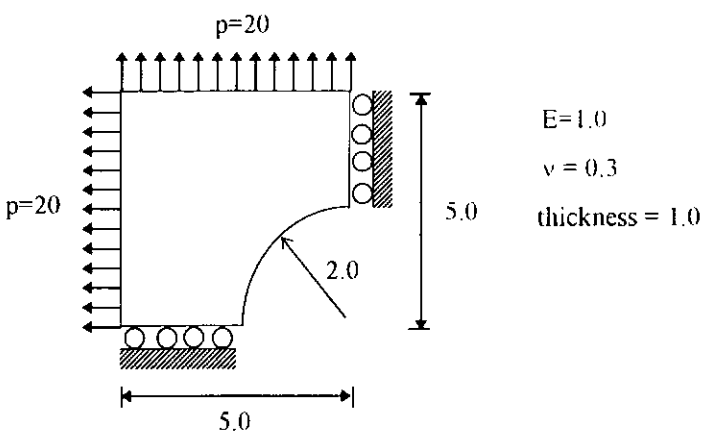
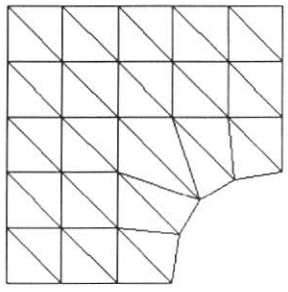
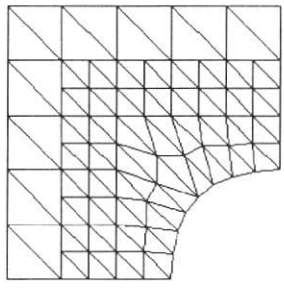


Figure 5.5. Square plate with circular hole domain problem.

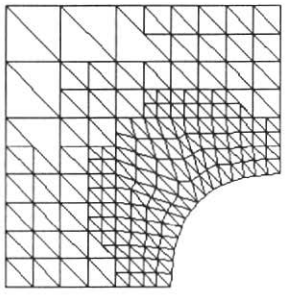
For this example a discretization tolerance $\eta = 2\%$ was adopted, and in Fig. 5.6 we can see the initial mesh and the sequence of five refined meshes obtained. It must be noted that due to the symmetry of this problem only one octant of the plate would be sufficient, but we take one quarter to verify the symmetry of the refinements.



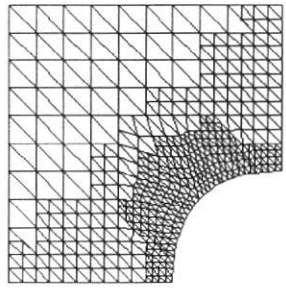
MESH 0 η^* = 16.41 %
DOF = 64



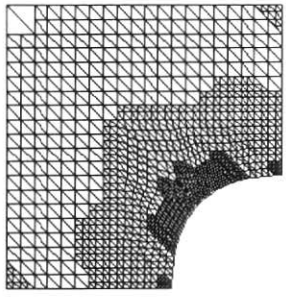
MESH 1 η^* = 10.72 %
DOF = 152



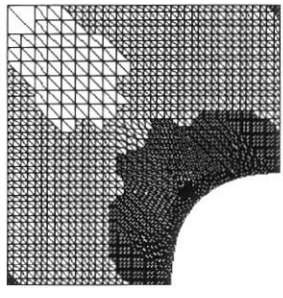
MESH 2 η^* = 6.33 %
DOF = 376



MESH 3 η^* = 3.96 %
DOF = 878



MESH 4 η^* = 2.43 %
DOF = 2140



MESH 5 η^* = 1.53 %
DOF = 5330

Figure 5.6. Square plate with a hole. Side flux averaging error estimator.

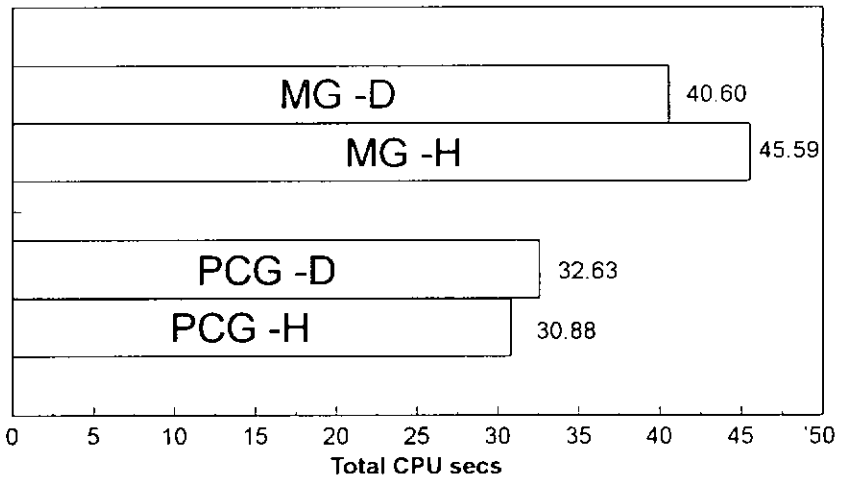


Figure 5.7. Square plate with a hole problem CPU time for fine mesh.

Table 5.3. Square plate problem - Number of iterations on fine mesh of each level.

	ITERATIONS ON FINE MESH OF EACH LEVEL			
MESH (DOF)	PCG - H	PCG - D	MG - H	MG - D
1 (152)	11	26	11 (2)	11 (2)
2 (376)	12	40	11 (2)	10 (1)
3 (878)	13	29	11 (2)	8 (1)
4 (2140)	11	15	8 (1)	7 (1)
5 (5330)	8	7	6 (1)	6 (1)
TOTAL TIME	30.88 secs.	32.63 secs.	45.59 secs.	40.60 secs.

Table 5.4. Square plate problem - Memory storage for fine mesh of each level.

	MEMORY STORAGE (Bytes)					
MESH (DOF)	PCG		MG - H		MG - D	
1 (152)	31856	(1.00)	34096	(1.07)	35824	(1.12)
2 (376)	73240	(1.00)	80280	(1.10)	85304	(1.16)
3 (878)	165790	(1.00)	185550	(1.12)	198942	(1.20)
4 (2140)	397916	(1.00)	456668	(1.15)	494604	(1.24)
5 (5330)	983858	(1.00)	1155906	(1.17)	1263250	(1.28)

In Figure 5.7 the CPU time spent for each solver is shown. We can see that the both multigrid algorithms were slower than the PCG solvers, and the hierarchical PCG was the fastest. It must be noted that the fact that the PCG-D solver take less time than the multigrid algorithms can be due to the characteristics of the problem whose solution is smooth and to the good approximations of the previous solutions on the coarse meshes. Note that in this example the MG-D solver was more efficient than the MG-H indicating that the more elaborated preconditioner has almost no effect on the solution. Also note that the number of iterations of the PCG-H solver and the number of multigrid V-cycles are almost constant for each mesh.

In Table 5.4 the memory requirements of each solver are shown. As in the previous example we note that the requirement of storage of the multigrid solvers increases exponentially with the degrees of freedom, and in Fig. 5.8 we can see the variation of memory storage requirements of each solver during the adaptive process.

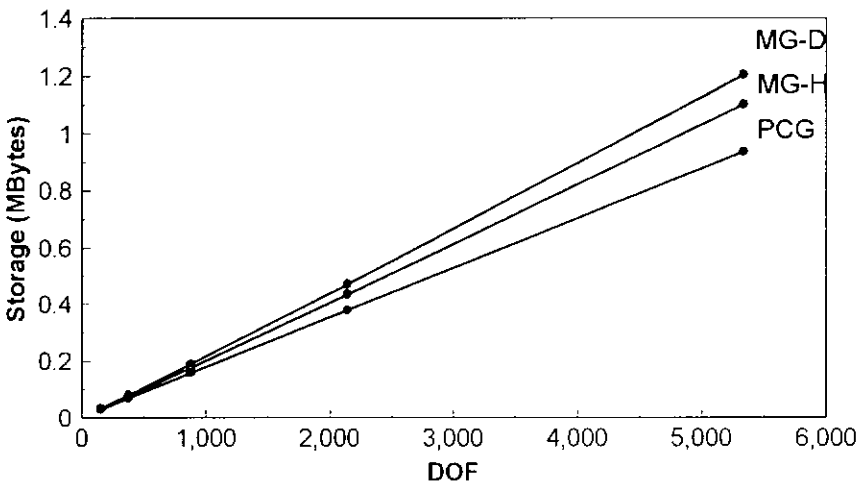


Figure 5.8. Square plate problem - Memory storage.

As in the previous example we made a comparison of the convergence rate of the relative energy norm of the increment $\| x^{(k+1)} - x^{(k)} \|_K / \| x^{(k+1)} \|_K$ and the error in the energy norm of the solution $\| x^{(k)} - x_h \|_K / \| x_h \|_K$. To compute the exact numerical value x_h we iterate till obtain a stabilized solution in the energy norm. For this example the value computed for the energy norm of the solution on the finer mesh was $\| x_h \|_K$ = 142.6348559856.

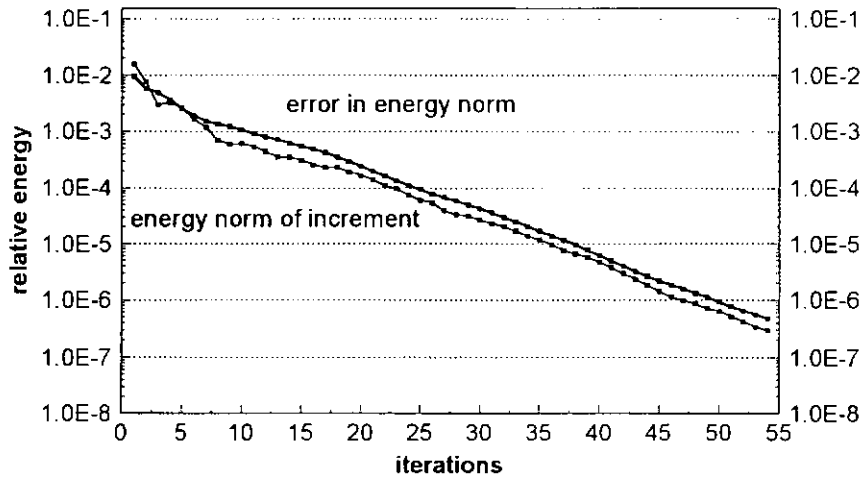


Figure 5.9. Comparison between error in energy norm and energy norm of increment.

As in the previous example we can see that both measures are very similar. The comparisons were made using the PCG-H solver on the final mesh.

5.2.3 Wrench problem.

We consider a wrench subject to uniform tractions concentrated on the extreme. Geometry, essential and natural boundary conditions are given in Fig. 5.6.

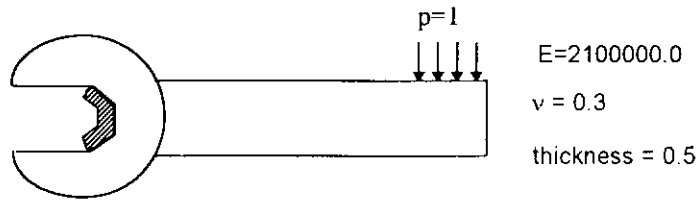


Figure 5.10. Wrench domain problem.

For this example a discretization tolerance η = 7% was adopted, and in Fig. 5.11 we can see the sequence of six refined meshes obtained from the initial coarse mesh which is shown in Fig. 5.12.

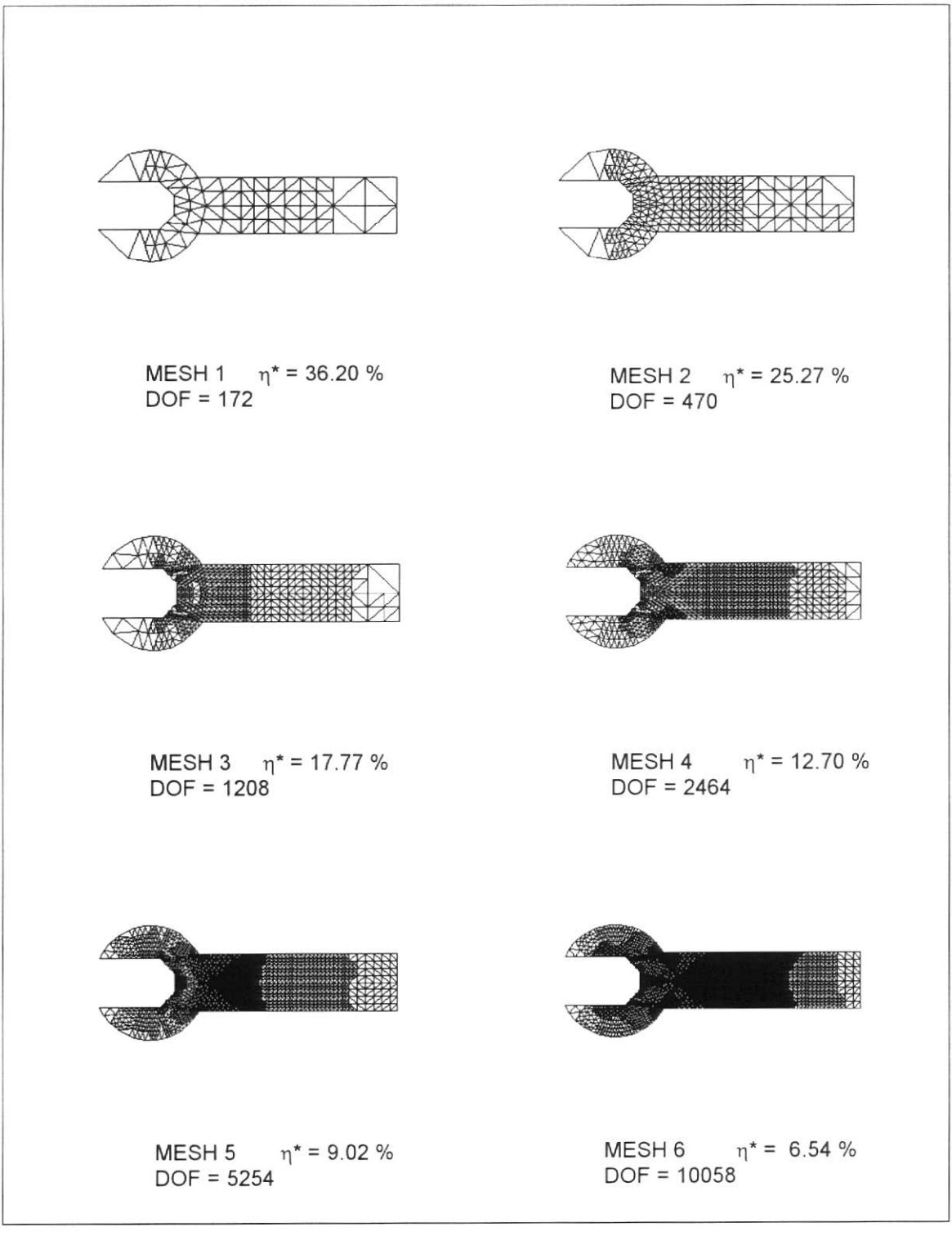


Figure 5.11. Wrench problem. Side flux averaging error estimator.

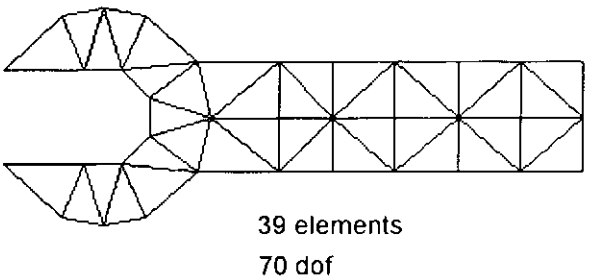


Figure 5.12. Wrench problem initial mesh.

In Figure 5.13 the CPU time spent for each solver is shown. We can see that the diagonal PCG was the slowest of all the algorithms, and the hierarchical PCG was the fastest. Both multigrid algorithms were slower than the hierarchical PCG. It must be noted that the fact that the PCG-D solver take more time than the multigrid algorithms can be due to the characteristics of the problem whose solution has several regions with singularities which are not well approximated by the previous solutions on the coarse meshes. Note that in this example the MG-D solver was more efficient than the MG-H indicating that the more elaborated preconditioner has almost no effect on the solution. Also note that the number of iterations of the PCG-H solver and the number of multigrid V-cycles for the MG-H solver are almost constant for each mesh.

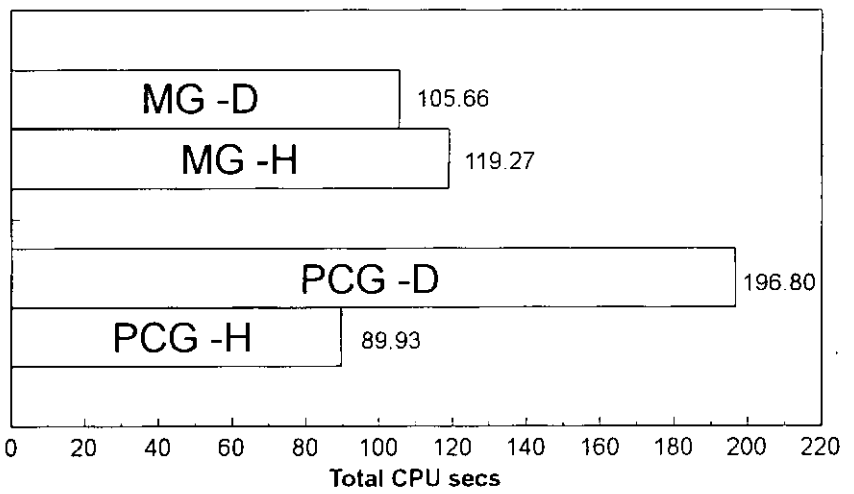


Figure 5.13. Wrench problem CPU time for fine mesh.

Table 5.5. Wrench problem - Number of iterations on fine mesh of each level.

	ITERATIONS ON FINE MESH OF EACH LEVEL			
MESH (DOF)	PCG - H	PCG - D	MG - H	MG - D
1 (172)	14	55	17 (2)	29 (3)
2 (470)	18	101	18 (2)	21 (3)
3 (1208)	19	177	16 (2)	18 (2)
4 (2464)	19	81	13 (2)	11 (2)
5 (5254)	19	50	11 (2)	10 (1)
6 (10058)	19	70	9 (1)	8 (1)
TOTAL TIME	89.93 secs.	196.80 secs.	119.27 secs.	105.66 secs.

In Table 5.6 the memory requirements of each solver are shown. As in the previous examples we note that the requirement of storage of the multigrid solvers increases exponentially with the degrees of freedom, and in Fig. 5.14 we can see the variation of memory storage requirements of each solver during the adaptive process.

Table 5.6. Wrench problem - Memory storage for fine mesh of each level.

	MEMORY STORAGE (Bytes)					
MESH (DOF)	PCG		MG - H		MG - D	
1 (172)	34118	(1.00)	36614	(1.07)	38550	(1.13)
2 (470)	89672	(1.00)	97656	(1.09)	103528	(1.15)
3 (1208)	224960	(1.00)	248608	(1.11)	265264	(1.18)
4 (2464)	456076	(1.00)	521964	(1.14)	564764	(1.24)
5 (5254)	966092	(1.00)	1136444	(1.18)	1242636	(1.29)
6 (10058)	1847306	(1.00)	2254458	(1.22)	2498266	(1.35)

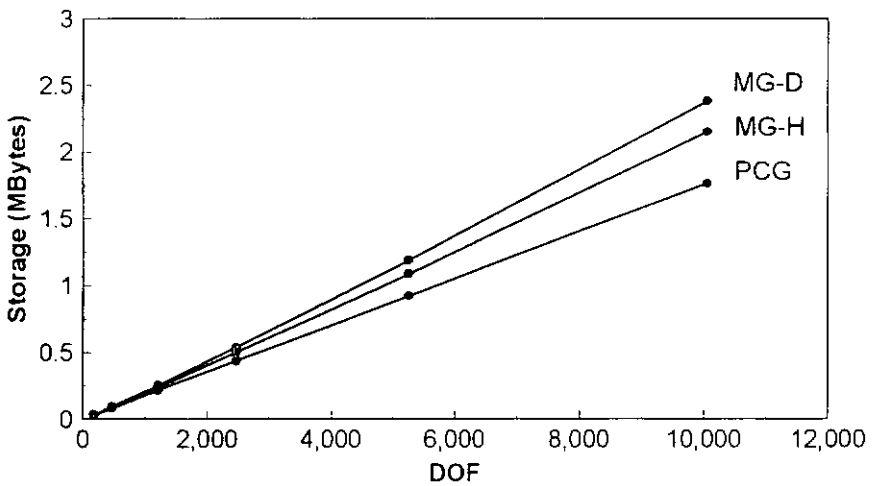


Figure 5.14. Wrench problem - Memory storage.

As in the previous examples we made also a comparison of the convergence rate of the relative energy norm of the increment and the error in the energy norm of the solution. Again we can see in figure 5.15 that both measures are almost identical.

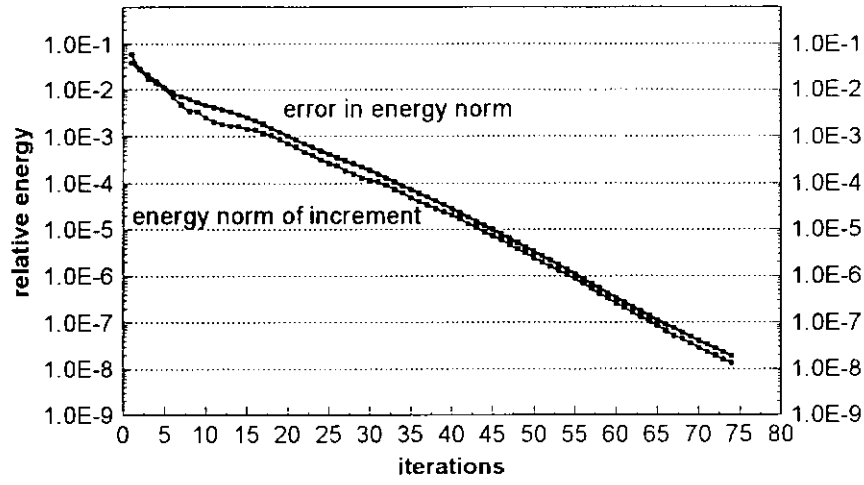


Figure 5.15. Comparison between error in energy norm and energy norm of increment.

5.3. Numerical Examples of Non-linear Adaptive Analysis

In the next sections some examples of adaptivity for bidimensional elasticity with geometrical nonlinearities are shown. In all the examples we use the modified Newton-Raphson method with a tolerance ε_0 =0.00001 for the convergence criteria in displacements, and the linear equations are solved with the PCG-H solver.

5.3.1. Non-linear Bending of a Cantilever Beam

We analyze a cantilever beam loaded by a bending moment at the extreme. The geometry, loads, boundary conditions are shown in Fig. 5.16.

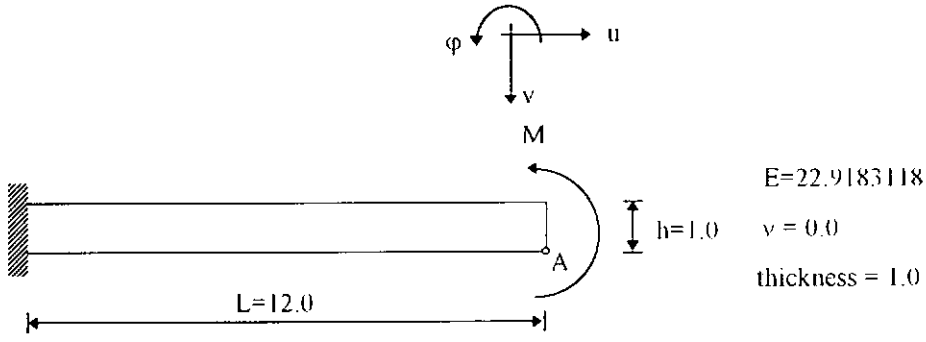


Figure 5.16. Cantilever beam loaded by a bending moment.

We consider large deflections and rotations but small deformations. For this case an analytic solution exists from beam theory [82] which express the extreme displacements u_L, v_L, φ_L as:

$$u_L = L\left(\frac{1}{2\pi\lambda}sin(2\pi\lambda) - 1\right) \tag{5.2.a}$$

$$v_L = \frac{L}{2\pi\lambda}\left(1 - cos(2\pi\lambda)\right) \tag{5.2.b}$$

$$\varphi_L = 2\pi\lambda \tag{5.2.c}$$

where

$$\lambda = M / M_0 \tag{5.3.a}$$

$$M_0 = 2\pi EI / L \tag{5.3.b}$$

and E is the elasticity modulus, I is the inertia of the cross section and L is the length of the beam. In particular for the bottom point A (see Fig.5.16) the horizontal and vertical displacements are:

$$u_A = u_L + \frac{h}{2}sin(2\pi\lambda) \tag{5.4.a}$$

$$v_A = v_L - \frac{h}{2}\left(1 - cos(2\pi\lambda)\right) \tag{5.4.b}$$

where *h* is the heigth of the cross section.

For small deflections these displacements can be approximated as

$$u_A \approx -\frac{h}{2}\varphi_L = \frac{M\,L\,h}{EI\;2} \tag{5.5.a}$$

$$v_A \approx v_L = \frac{M\,L^2}{2\,EI} \tag{5.5.b}$$

We use the displacements of this point as the reference values for comparisons.

In Fig. 5.17 we can see the coarse mesh of the beam and the end moment which is modeled by an equivalent linear distribution of normal stress.

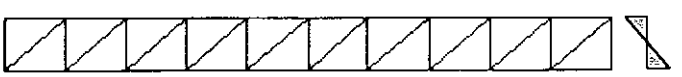


Figure 5.17. Cantilever beam . Initial mesh.

We made previously an adaptive linear analysis and after six adaptive refinements we obtain the history shown in Fig. 5.18 for the vertical displacements at the extreme

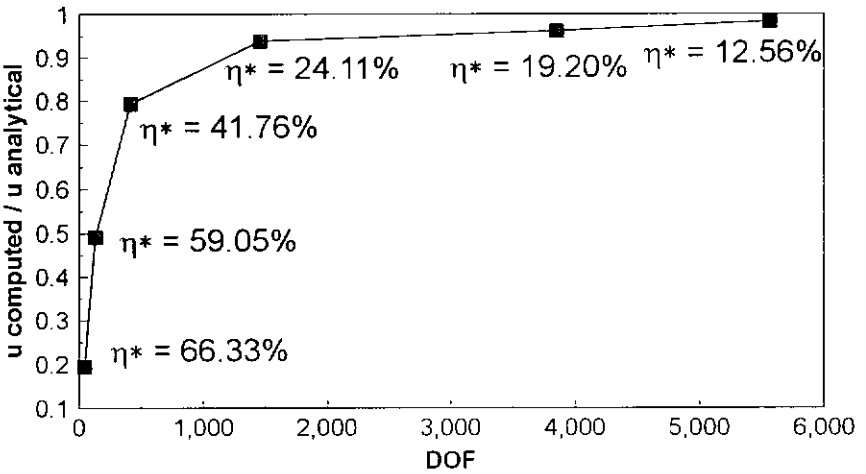


Figure 5.18. Cantilever beam . Bottom vertical deflection.

Also, in this figure are indicated the computed error estimates. The computed error $\eta^* = 24.11\%$ correspond to a quasi-uniform discretization of the beam with 1458 degrees of freedom (DOF). A detail of this mesh is shown in Fig. 5.19. The computed error $\eta^* = 12.56\%$ correspond also to a quasi-uniform discretization of the beam with 5560 degrees of freedom (DOF). A detail of this mesh is shown in Fig. 5.20.

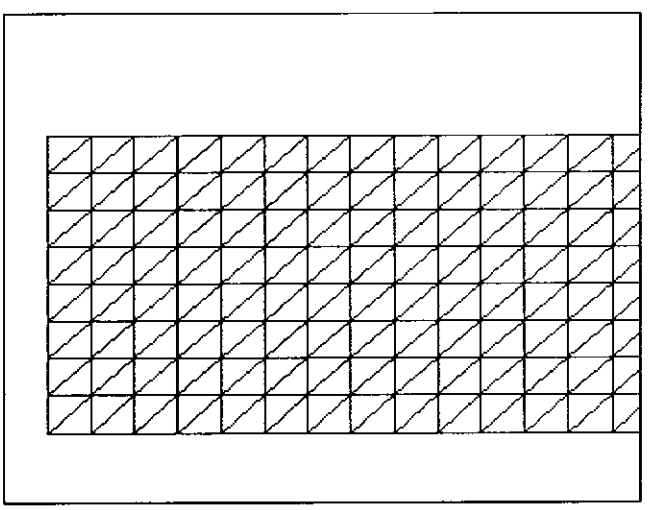


Figure 5.19. Cantilever beam . Detail of mesh of 1458 DOF.

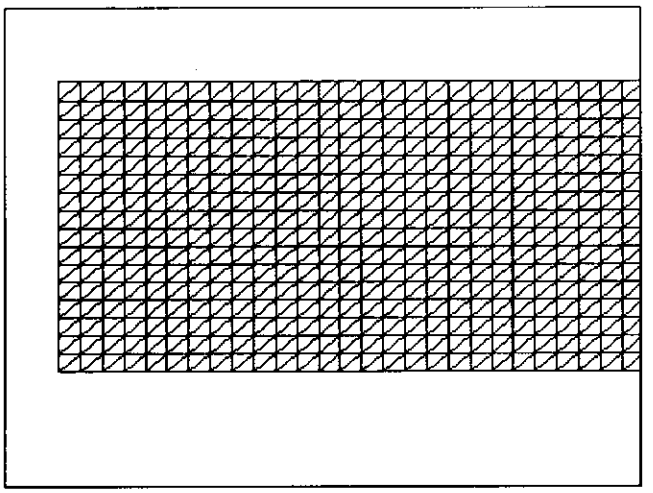


Figure 5.20. Cantilever beam . Detail of mesh of 5560 DOF.

We made an adaptive incremental non-linear analysis with six increments of load factor $\Delta\lambda$=0.00125 and for two discretization tolerances, 13% and 25%, initiating the adaptive process with the coarse mesh of Fig. 5.17. For this particular example the meshes obtained during the initial refinement, which correspond to the linear analysis, satisfied the error criteria for all the non-linear analysis. An averaged value of seven Newton-Raphson (NR) iterations were needed in each step to reach equilibrium for the mesh of 1458 DOF, and an averaged value of eigth NR iterations were needed with the mesh of 5560 DOF. The results for the horizontal and vertical deflection of the point A can be seen in Figs. 5.21 and 5.22 respectively.

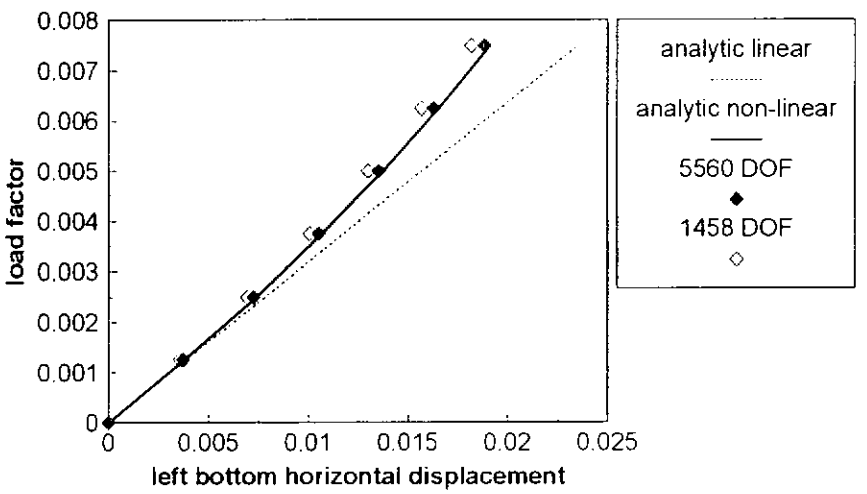


Figure 5.21. Cantilever beam . Horizontal displacement point A.

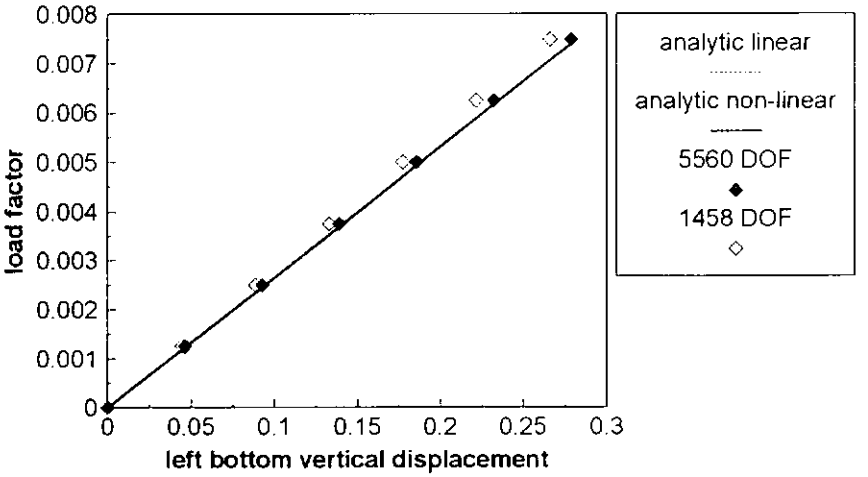


Figure 5.22. Cantilever beam . Vertical displacement point A.

We can see in this example that there are no appreciable differences between the results for meshes of 1458 DOF and 5560 DOF. But the differences in execution time are considerable, as can be seen in Fig. 5.23.

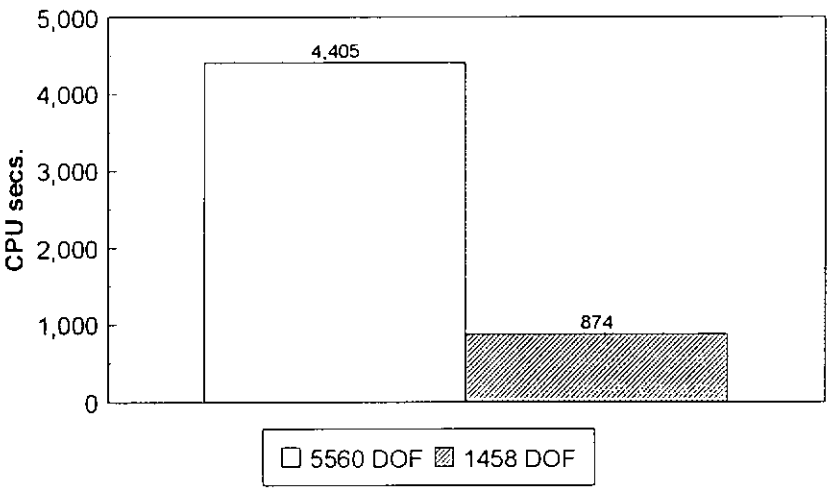


Figure 5.23. Cantilever beam . Comparison execution time.

5.3.2. Clamped Arch Under Uniform Pressure

In this example we analize a clamped circular arch under uniform pressure. The geometry (not in scale), loads, boundary conditions and the initial mesh are shown in Fig. 5.24.

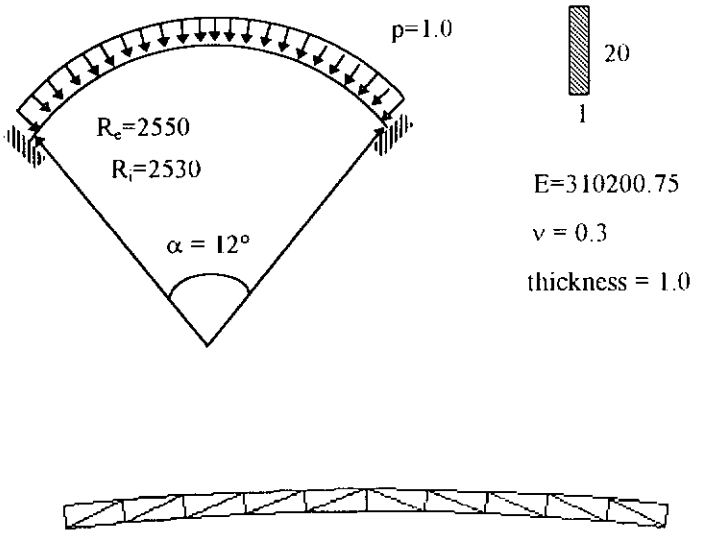


Figure 5.24. Clamped arch under uniform pressure.

We consider large deflections and rotations but small deformations. For this example we made an adaptive incremental non-linear analysis with ten increments of load factor $\Delta p=1.0$ and for a discretization tolerance of 50%, initiating the adaptive process with the coarse mesh of Fig. 5.24. We use a high value of the discretization tolerance since the purpose of this example is to show the variation of the mesh along the incremental analysis. It must be noted that if we were using a low tolerance the initial mesh is uniformly refined in the first step and is not modified in the subsequent steps.

In Fig. 5.25 we can see the evolution of the midspan vertical deflection

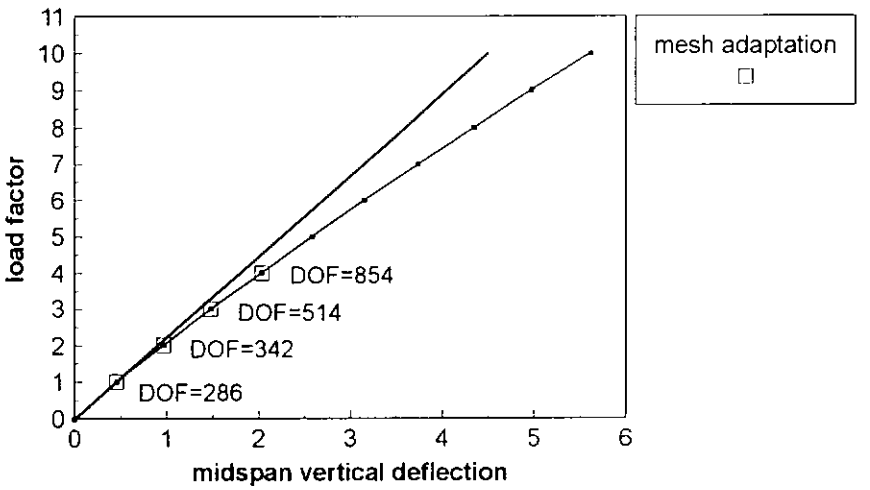


Figure 5.25. Clamped arch. Evolution of midspan vertical deflection.

In Fig. 5.26 to 5.29 the four final meshes obtained in each step are shown.

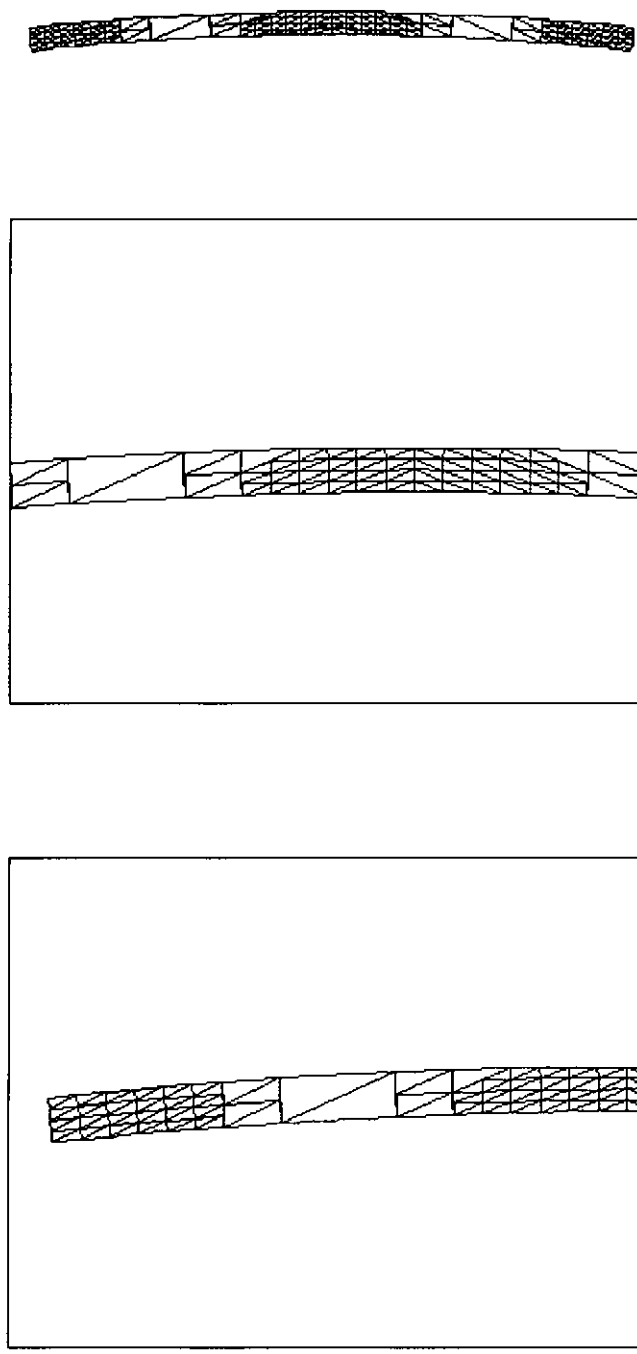


Figure 5.26. Clamped arch. First adapted mesh. DOF = 286.

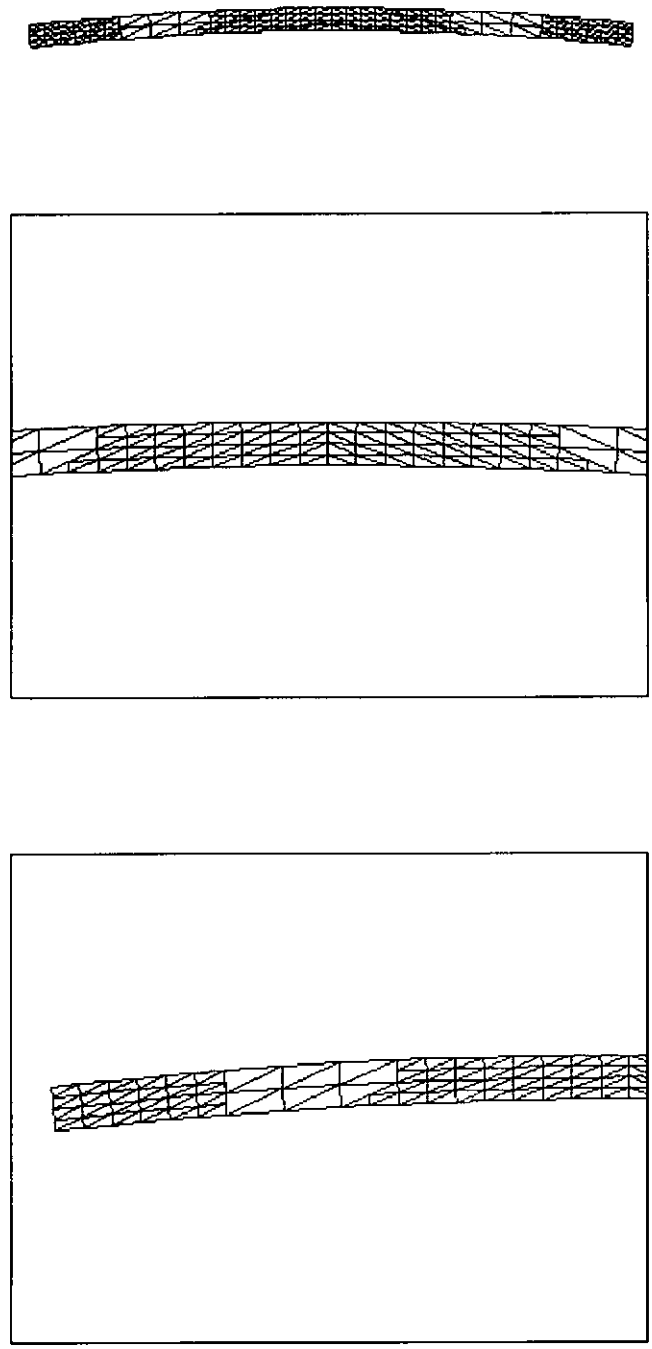


Figure 5.27. Clamped arch. Second adapted mesh. DOF = 342.

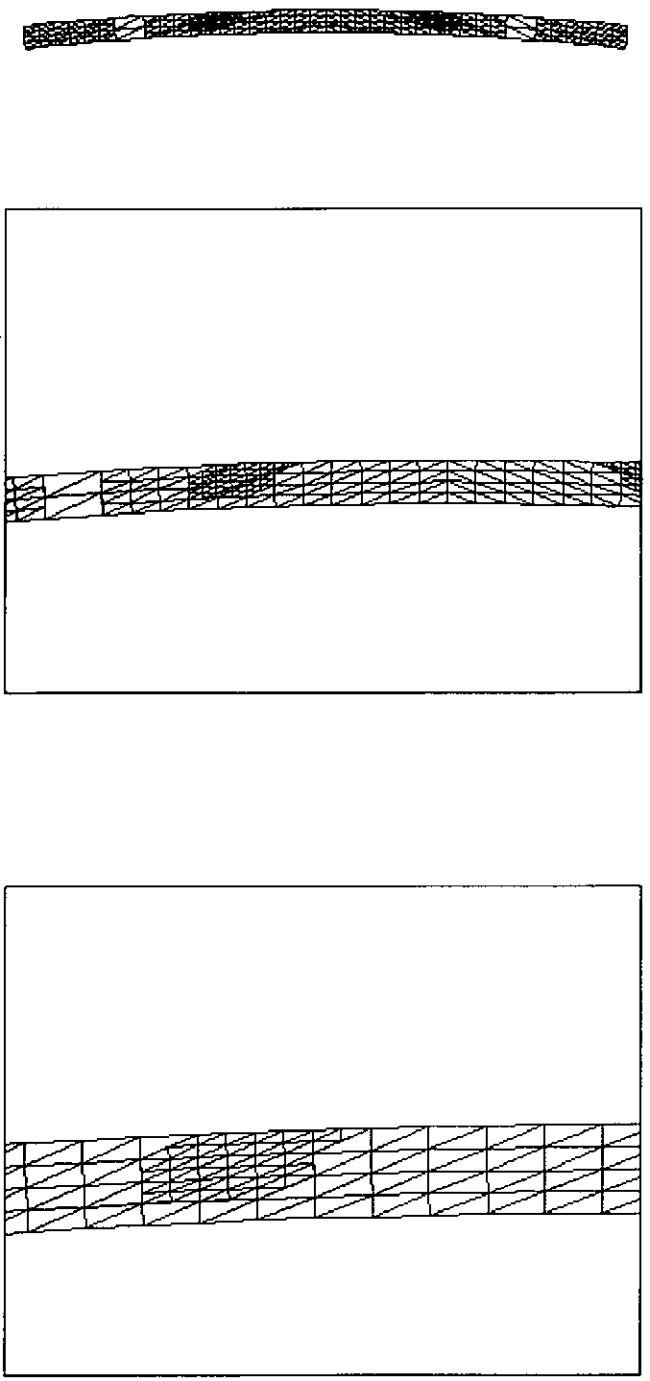


Figure 5.28. Clamped arch. Third adapted mesh. DOF = 514.

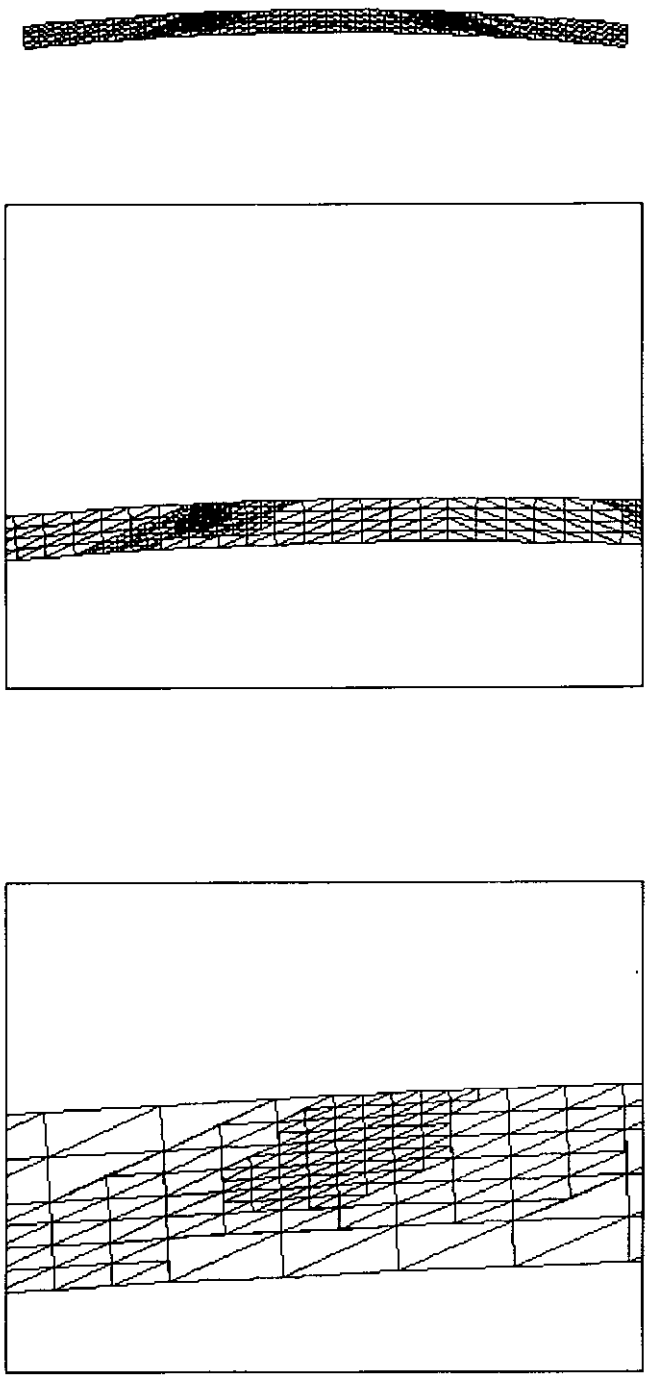


Figure 5.29. Clamped arch. Fourth adapted mesh. DOF = 826.

5.3.3. Plane with a Crack Under Traction

In this example we analize a plane with constant loads at two opposite sides and a crack in the midle, due to symmetry only one quarter of the plane is considered Geometry, loads, boundary conditions and the initial mesh are shown in Fig. 5.30.

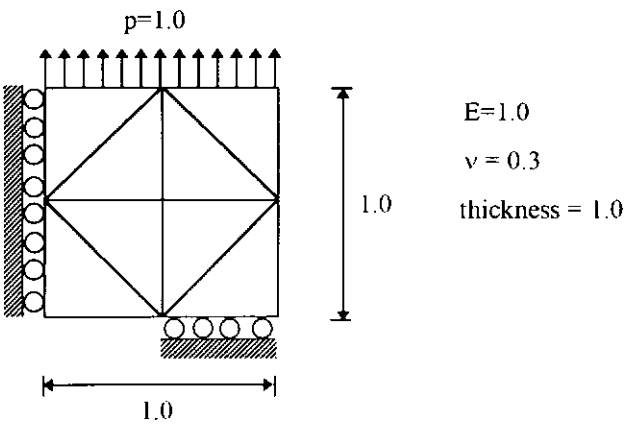


Figure 5.30. Plane under traction with a crack in the midle.

We consider large deflections and rotations but small deformations. For this example we made an adaptive incremental non-linear analysis with ten increments of the load factor $\Delta p=0.01$ and for a discretization tolerance of 17% , initiating the adaptive process with the coarse mesh of Fig. 5.30. In Fig. 5.31 we can see the evolution of the crack opening. An averaged number of five Newton-Raphson iterations were needed for step, and two mesh adaptations were made at the first and second step, whose respective meshes are shown in Figs. 5.32 and 5.33. In Fig. 5.34 we can see the final deformed configuration.

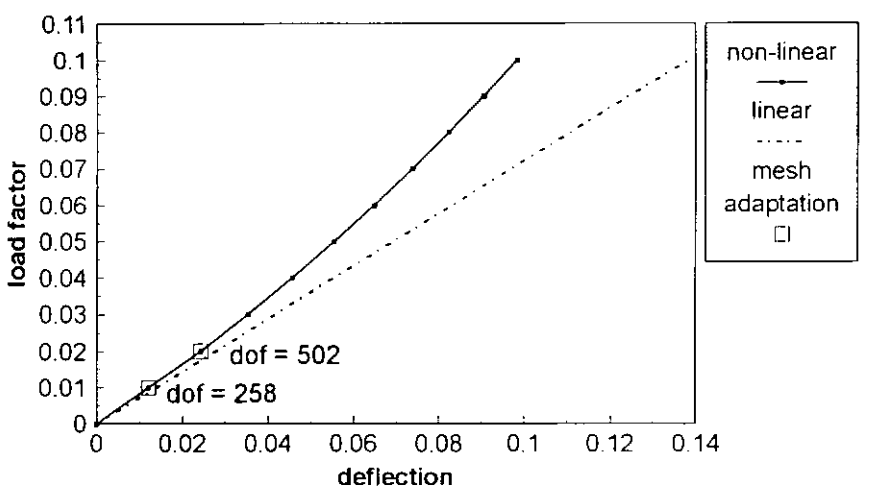


Figure 5.31. Plane with a crack in the midle. Crack opening evolution.

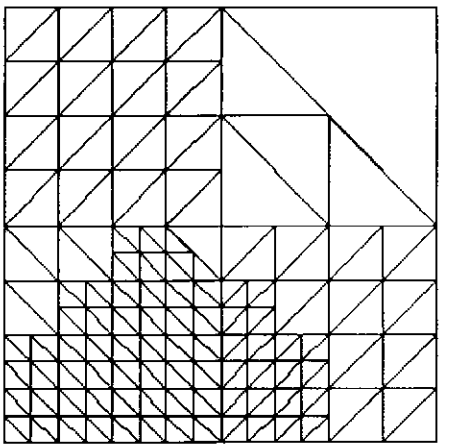


Figure 5.32. Plane with a crack in the midle. DOF = 258.

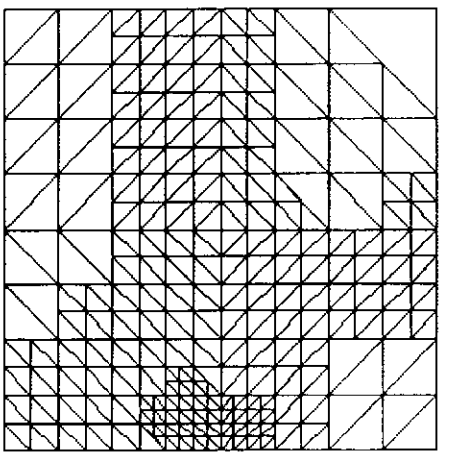


Figure 5.33. Plane with a crack in the midle. DOF = 502.

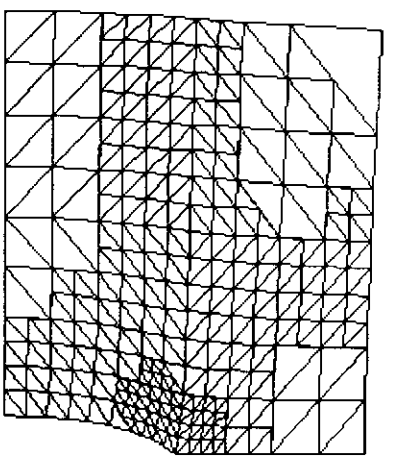


Figure 5.34. Plane with a crack in the midle. Deformed configuration for p=0.1

Chapter 6

Conclusions

6.1 Thesis Contributions

In chapter 2, we presented an error estimator for linear triangles based on stress projections. We use local side coordinates to define normal and tangential fluxes on each side, which are averaged between adjacent elements. At loaded boundaries the smoothed gradient field can be enforced to adopt the averaged values of prescribed fluxes in a simple and natural manner. The consideration of prescribed fluxes appears to be a fundamental factor to obtain reliable error estimates, as it can be seen in the numerical example of the short cantilever beam.

Instead of averaging the stress components at mid-side points we have averaged the total forces acting on each side, since they must be equal for adjacent elements even if the stress field is discontinuous due to abrupt material or thickness changes. In this manner no distinction must be made between elements with different properties. It must be noted that this treatment of the discontinuities can be useful in other areas, such as flow through porous media, where the geological description of the subsurface, typically generated using some combination of measured data and geostatistics, often display very large variations in physical properties [83].

The stress predictions are only continuous at mid-side points. This lack of continuity serves as a measure of the quality of the approximation which complements the more abstract measure of the error in the energy norm. From the numerical comparisons with other error estimators we can conclude that the performance of the presented error estimator is very good.

In chapter 3 the data structure for the adaptive code was presented. It is organized by levels and consists of the traditional finite element data structure supplemented by additional arrays of connectivity information.

In chapter 4 the iterative multivel solvers were presented. Also a convergence criteria based in energy norms was presented and its relation with adaptive refinements was numerically analyzed. It was shown that the choice of the numerical tolerance must be based on the quality of the error estimator and not on the number of degrees of freedom of the adapted meshes. It must be noted that the energy criteria has the advantage of being non-dimensional.

In chapter 5 numerical comparisons were made for the iterative solvers in linear adaptive problems. From the examples tested we can conclude that the hierarchical preconditioned conjugate gradients (PCG-H) was the most efficient of the algorithms. Multigrid solvers were slower than the PCG-H solver. The superiority of the hierarchical PCG solver over the multigrid solvers in all the examples tested is an important fact, since multigrid solvers need more storage than PCG solvers and the efficiency of multigrid solvers rely on coefficients which must be determined by numerical experiments, such as the numbers of relaxation steps ν_1, ν_2 on the intermediate meshes [63],[64].

Also in chapter 5 some examples of geometrical non-linear problems with adaptivity were tested. From the examples tested we can conclude that the adaptivity strategy was adequate for the treatment of the geometrical non-linearities. It must be noted that for prismatic structures, such as beams and arches, modeled by linear triangles a great number of elements are necessary due to the poor representation of *bending* states of these finite elements. Also for problems with moderately geometrical non-linearities the bending states have no significant changes during the incremental analysis, then the initial meshes, which are generally composed of a very large number of elements, are sufficiently refined for the rest of the incremental analysis and no further refinements are necessary.

6.2 Future Research

It must be noted that the side based error estimator is well suited for parallel codes with edge oriented data structure, then a natural continuation of this work is the implementation of this error estimator in parallel and vector machines.

The application of the error estimator to other finite elements such as tri-dimensional tetrahedrals, is direct, although some minimum modifications are needed in the data structure since in this case the averaged quantities are located on the faces of the elements.

A natural extension of the application of the flux averaging error estimator is in the field of computational fluid dynamics where it can be combined with adaptive *h*-refinement and de-refinement [84]. The fully nested property of these subdivision techniques can be exploited by the multilevel algorithms presented in chapter 4.

More tests are necessary to confirm the superiority of the hierarchical PCG over multigrid methods, especially on parallel adaptive codes. Also, a possible improvement of the hierarchical preconditioner can be obtained when combined with domain decomposition methods or substructuring techniques.

More nonlinear applications need to be implemented to assess the effectivity of the techniques here proposed, such as plasticity problems or contact problems, where the variation of the optimal mesh during the incremental analysis is more accentuated that in the examples that were shown here.

Appendix A

Coordinate Transformation

Consider a general linear coordinate transformation for displacements x

$$x = T\hat{x} \tag{A.1}$$

where $\hat{x}$ is the vector of transformed coordinates which can have less components than vector x and consequently the transformation matrix T is not necessary square.

The equilibrium equations in these two systems can be written as:

$$Kx = f \tag{A.2}$$

$$\hat{K}\hat{x} = \hat{f} \tag{A.3}$$

and we are interested in the relations between quantities of both systems. Since f and $\hat{f}$ describe the same resultant force, work done by the force during a prescribed virtual displacement must be independent of the coordinate system in which the work is computed. Let δx and $\delta\hat{x}$ be the same virtual displacement expressed in both systems, and which components are related by eqn.(A.1) as

$$\delta x = T\delta\hat{x} \tag{A.4}$$

then from the equality of the virtual work in both systems we have

$$\delta x^{\mathrm{T}} f = (\delta\hat{x}^{\mathrm{T}} T) f = \delta\hat{x}^{\mathrm{T}}\hat{f} \tag{A.5}$$

from which

$$\delta\hat{x}^{\mathrm{T}}(T^{\mathrm{T}} f - \hat{f}) = 0 \tag{A.6}$$

and since this expression must be valid for *any* virtual displacement, we obtain

$$\hat{f} = T^{\mathrm{T}} f \tag{A.7}$$

The, by substitution into eqn.(A.2) we have

$$\hat{K}\hat{x} = \hat{f} = T^{T} f = T^{T} K x = T^{T} K T \hat{x} \tag{A.8}$$

from which

$$\hat{K} = T^{T} K T \tag{A.9}$$

which express the relation between the stiffness matrices in both systems.

References

[1] Papadrakakis, M., Solving large-scale linear problems in solid and structural mechanics, in: M. Papadrakakis, ed., *Solving Large-scale Problems in Mechanics*, John Wiley & Sons Ltd, Chichester, England (1993).

[2] Briggs, W.L., *A Multigrid Tutorial*. SIAM, Philadelphia (1987).

[3] Bank, R.E. and Sherman, A.H., An adaptive, multilevel method for elliptic boundary value problems, *Computing*, **26**, 91-105 (1981).

[4] Rivara, M.C., Algorithms for refining triangular grids suitable for adaptive and multigrid techniques, *Int. J. Numer. Methods Eng.*, **20**, 745-756 (1984).

[5] Rivara, M.C., Design and data structure of fully adaptive, multigrid, finite-element software, *ACM Trans. Math. Software*, **10**, 242-264 (1984).

[6] Rheinboldt, W.C. and Mesztenyi, Ch.K., On a data structure for adaptive finite element mesh refinements, *ACM Trans. Math. Software*, **6**, 166-187 (1980).

[7] Devloo, Ph., Oden, J.T. and Stroboulis, T., Implementation of an adaptive refinement technique for the SUPG algorithm, *Comput. Meths. Appl. Mech. Eng.*, **61**, 339-358 (1987).

[8] Löhner, R., An adaptive finite element scheme for transient problems in CFD, *Comput. Meths. Appl. Mech. Eng.*, **61**, 323-338 (1987).

[9] Babuska, I., The problem of modeling the elastomechanics in engineering, *Comput. Meths. Appl. Mech. Eng.*, **82**, 155-182 (1990).

[10] Bathe, K.J., Lee, N.S. and Bucalem, M., On the use of hierarchical models in engineering analisys, *Comput. Meths. Appl. Mech. Eng.*, **82**, 5-26 (1990).

[11] Bathe, K.J., *Finite Elements Procedures in Engineering Analysis*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey (1982).

[12] Zienkiewicz O.C., and Taylor R.L., *The Finite Element Method*, 4th edn., Vol.1, McGraw-Hill, New York, (1989).

[13] Hughes, T.J.R., *The Finite Element Method*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey (1987).

[14] Babuska, I., and Rheinboldt, W.C., Error estimates for adaptive finite element computations, *SIAM J. Numer. Anal.*, **15 (4)**, 736-754 (1978).

[15] Babuska, I., and Rheinboldt, W.C., A-posteriori error estimates for the finite element method, *Int. J. Numer. Methods Eng.*, **12**, 1597-1615 (1978).

[16] Ainsworth, M. and Craig, A., A-posteriori error estimators in the finite element method, *Numer. Math.*, **60**, 429-463 (1992).

[17] Ainsworth, M. and Oden , J.T., A unified approach to a posteriori error estimation using element residual methods, *Numer. Math.*, **65**, 23-50 (1993).

[18] Babuska, I., Duran, R. and Rodriguez, R., Analysis of the efficiency of an a-posteriori error estimator for linear triangular finite elements, *SIAM J. Numer. Anal.*, **29 (4)**, 947-964 (1992).

[19] Babuska, I., and Yu, D., Asymptotically exact a-posteriori error estimator for biquadratic elements, *Finite Elements Anal. Design*, **3**, 341-354 (1987).

[20] Bank, R.E. and Weiser, A., Some a-posteriori error estimates for elliptic partial differential equations, *Math. Comp.*, **44**, 283-301 (1985).

[21] Durán, R., Muschietti, M.A. and Rodriguez R., On the asymptotic exactness of the error estimator for linear triangular elements, *Numer. Math.*, **59**, 78-88 (1991).

[22] Kelly, D.W., The self-equilibration of residuals and complementary a-posteriori error estimates in the finite element method, *Int. J. Numer. Methods Eng.*, **20**, 1491-1506 (1984).

[23] Ladeveze, P. and Leguillon, Error estimation procedure in the finite element method and applications, *SIAM J. Numer. Anal.*, **20**, 485-509 (1983).

[24] Oden, J.T., Demkowicz, L., Rachowicz, W. and Westermann, T.A., Toward a universal h-p adaptive finite element strategy, Part 2: a-posteriori error estimates, *Comput. Meths. Appl. Mech. Eng.*, 77, 113-180 (1989).

[25] Verfürth, R., A-posteriori error estimators for the Stokes equation, *Numer. Math.*, **55**, 309-325 (1989).

[26] Szabo, B.A., Mesh design for the p-version of the finite element method, *Comput. Meths. Appl. Mech. Eng.*, **55**, 181-197 (1986).

[27] Demkowicz, L., Devloo, P. and Oden, J.T., On h-type mesh-refinement strategy based on minimization of interpolation error, *Comput. Meths. Appl. Mech. Eng.*, **53**, 67-89 (1985).

[28] Eriksson, K. and Johnson C., An adaptive finite element method for linear elliptic problems, *Math. Comp.*, **50**, 361-383 (1988).

[29] Zienkiewicz, O.C., and Zhu, J.Z., A simple error estimator and adaptive procedure for practical engineering analysis, *Int. J. Numer. Methods Eng.*, **24**, 337-357 (1987).

[30] Zienkiewicz, O.C., and Zhu, J.Z., The superconvergent patch recovery and a posteriori error estimates. Part 1: The recovery technique, *Int. J. Numer. Methods Eng.*, **33**, 1331-1364 (1992).

[31] Zienkiewicz, O.C., and Zhu, J.Z., The superconvergent patch recovery and a posteriori error estimates. Part 2: Error estimates and adaptivity, *Int. J. Numer. Methods Eng.*, **33**, 1365-1382 (1992).

[32] Wiberg, N.E., Abdulwahab, F. and Ziukas, S., Enhanced superconvergent patch recovery incorporating equilibrium and boundary conditions, *Int. J. Numer. Methods Eng.*, **37**, 3417-3440 (1994).

[33] Blacker, T. and Belytschko, T., Superconvergent patch recovery with equilibrium and conjoint interpolant enhancements, *Int. J. Numer. Methods Eng.*, **37**, 517-546 (1994).

[34] Babuska, I., Strouboulis, T., Upadhyay, C.S., Gangaraj, S.K. and Copps, K., Validation of a posteriori error estimators by numerical approach, *Int. J. Numer. Methods Eng.*, **37**, 1073-1123 (1994).

[35] Strouboulis, T. and Haque, K.A., Recent experiences with error estimation and adaptivity, Part I: review of error estimators for scalar elliptic problems, *Comput. Meths. Appl. Mech. Eng.*, **97**, 399-436 (1992).

[36] Strouboulis, T. and Haque, K.A., Recent experiences with error estimation and adaptivity, Part II: error estimation for h-adaptive approximations on grids of triangles and quadrilaterals, *Comput. Meths. Appl. Mech. Eng.*, **100**, 359-430 (1992).

[37] Stein, E., and Ahmad, R., On stress computation in finite elements models based upon displacement approximations, *Comput. Meths. Appl. Mech. Eng.*, **4**, 81-96 (1974).

[38] Oden, J.T., and Brauchli, H.T., On the calculation of consistent stress distribution in finite elements calculations, *Int. J. Numer. Methods Eng.*, **3**, 317-325 (1971).

[39] Oden, J.T., and Reddy, J.N., Note on an approximate method for computing consistent conjugate stresses in elastic finite elements, *Int. J. Numer. Methods Eng.*, **6**, 55-61 (1973).

[40] Hinton, E. and Campbell, J.S., Local and global smoothing of discontinuos finite element functions using a least squares method, *Int. J. Numer. Methods Eng.*, **8**, 461-480 (1974).

[41] Barlow, J., Optimal stress locations in finite element models, *Int. J. Numer. Methods Eng.*, **10**, 243-251 (1976).

[42] Mackinnon, R.J. and Carey, G.F., Superconvergent derivatives: ATaylor series analysis, *Int. J. Numer. Methods Eng.*, **28**, 489-509 (1989).

[43] Zienkiewicz, O.C., and Zhu, J.Z., The SPR recovery and boundaries, *Int. J. Numer. Methods Eng.*, **37**, 3195-3201 (1994).

[44] Babuska, I., and Rheinboldt, W.C., Analysis of optimal finite-element meshes in R^1, *Math. Comp.*, **33**, 435-463 (1979).

[45] Diaz, A.R., Kikuchi, N. and Taylor, J.E., A method of grid optimization for finite element methods, *Comput. Meths. Appl. Mech. Eng.*, **41**, 29-45 (1983).

[46] Zhu, J.Z. and Zienkiewicz, O.C., Adaptive techniques in the finite element method, *Commun. Appl. Numer. Methods*, **4**, 197-204 (1988).

[47] Lo, S.H., A new mesh regeneration scheme for arbitrary planar domains, *Int. J. Numer. Methods Eng.*, **21**, 1403-1426 (1985).

[48] Peraire, J., Vahdati, M., Morgan, K. and Zienkiewicz, O.C., Adaptive remeshing for compressible flow computations, *J. Comp. Phys.*, **72**, 449-466 (1987).

[49] Jin, H. and Wiberg, N.E., Two dimensional mesh generation, adaptive remeshing and refinement, *Int. J. Numer. Methods Eng.*, **29**, 1501-1526 (1990).

[50] Lo, S.H., Automatic mesh generation and adaptation using contours, *Int. J. Numer. Methods Eng.*, **31**, 689-707 (1991).

[51] Oden, J.T., Demkowicz, L., Rachowicz, W. and Hardy, O., Toward a universal h-p adaptive finite element strategy, Part 1: Constrained approximation and data structure, *Comput. Meths. Appl. Mech. Eng.*, 77, 79-112 (1989).

[52] Fung, Y. C., *Foundations of Solid Mechanics*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey (1965).

[53] Crisfield, M.A., *Non-Linear Finite Element Analysis of Solids and Structures*, John Wiley and Sons Ltd, (1991)

[54] Zienkiewicz, O.C., and Zhu, J.Z., Error estimates and adaptive refinements for plate bending problems, *Int. J. Numer. Methods Eng.*, **28**, 2839-2853 (1989).

[55] Levine, N., Stress sampling points for linear triangles in the finite element method, *Numer. Anal.*, 5, 407-427 (1985).

[56] Ainsworth, M., Zhu, J.Z., and Craig ,A.W. and Zienkiewicz, O.C., Analysis of the Zienkiewicz-Zhu a-posteriori error estimator in the finite element method, *Int. J. Numer. Methods Eng.*, **28**, 2161-2174 (1989).

[57] Babuska, I., Planck, L. and Rodriguez, R., Basic problems of a posteriori error estimation, *Comput. Meths. Appl. Mech. Eng.*, **101**, 97-112 (1992).

[58] Carey, G.F., Chow, S.S. and Seager, M.K., Approximate boundary-flux calculations, *Comput. Meths. Appl. Mech. Eng.*, **50**, 107-120 (1985).

[59] Mizukami, A., A mixed finite element method for boundary flux computation, *Comput. Meths. Appl. Mech. Eng.*, **57**, 239-243 (1986).

[60] Fish, J., Markolefas, S., Guttal, R. and Nayak, P., On adaptive multilevel superposition of finite element meshes for linear elastotastics, *Applied Numerical Mathematics*, **14**, 135-164 (1994).

[61] Babuska, I. and Rodriguez, R., The problem of the selection of an a posteriori error indicator based on smoothening techniques, *Int. J. Numer. Methods Eng.*, **36**, 539-567 (1993).

[62] Misra, H. and Parsons, I.D., Adaptive finite element multigrid methods on parallel computers, in: B.H.V. Topping and M. Papadrakakis, eds., *Advances in Parallel and Vector Processing for Structural Engineering*, Civil-Comp Ltd, Edinburgh, Scotland (1994).

[63] Parsons, I.D. and Hall, J.F., The multigrid method in solid mechanis: part I - algorithms description and behaviour, *Int. J. Numer. Methods Eng.*, **29**, 719-738 (1990).

[64] Parsons, I.D. and Hall, J.F., The multigrid method in solid mechanis: part II - practical applications, *Int. J.Numer.Methods Eng.*, **29**, 739-754 (1990).

[65] Hackbush, W., *Multi-Grid Methods and Applications*, Springer-Verlag, Berlin (1985).

[66] McCormick, S.F., *Multigrid Methods*, SIAM, Philadelphia (1987).

[67] Bank, R.H., Dupont, T., and Yserentant, H., The hierarchical basis multigrid method, *Numerische Mathematik*, **52**, 427-458 (1988).

[68] Bank, R.E., and Yserentant, H., Some remarks on the hierarchical basis multigrid method, in: T.F. Chan, R. Glowinski, J. Periaux and O.B. Widlund, eds., *Proc. of the Second Int. Symposium on Domain Decomposition Methods*, SIAM, Philadelphia, 140-146, (1989).

[69] Yserentant, H., On the multi-level splitting of finite element spaces for indefinite elliptic boundary value problems, *SIAM J. Num. Anal.*, **23**, 581-595 (1986).

[70] Yserentant, H., Hierarchical bases give conjugate gradient type methods a multigrid speed of convergence, *Applied Mathematics and Computation*, **19**, 347-358 (1986).

[71] Coutinho, A.L.G.A., Alves, J.L.D., Ebecken, N.F.F. and Devloo, P.R., Two-level hierarchical preconditioners for finite element equations, in: I.D. Parsons and B. Nour-Omid, eds., *ASME Winter Annual Meeting*, Atlanta, 47-55 (Dec. 1991).

[72] Barra, L.P.S., Coutinho, A.L.G.A., Telles, J.C.F. and Mansur, W.J., Multi-level hierarchical preconditioners for boundary element systems, *Eng. Anal. Bound. Elem.*, **12**, 103-109 (1993).

[73] Hestenes, M.R., and Stiefel, E., Methods of conjugate gradients for solving linear systems, *J. Res. Nat. Bur. Stand.*, **49**, 409-436 (1952).

[74] Reid, J.K., On the method of conjugate gradients for the solution of large sparse systems of linear equations, in J.K. Reid, ed., *Large Sparse Sets of Linear Equations*, Academic Press, New York, 231-254 (1970).

[75] Brandt, A., Multi-level adaptative adaptive solutions to boundary-value problems, *Math. Comp.*, **31**, 333-390 (1977).

[76] Jouglard, C.E., and Coutinho, A.L.G.A., A comparison of iterative multi-level finite element solvers, *Comp.& Struc.*, submitted for publication (1996).

[77] Papadrakakis, M., and Bibilis, G.P., Solution techniques for the p-version of the adaptive finite element method, *Int. J. Numer. Methods Eng.*, **37**, 1413-1431 (1994).

[78] Silva, R.S., Garcia, E.L.M., and Coutinho A.L.G.A., An algorithm to control the algebraic error in adaptive refinement (in Portuguese), in *Proc. of the XII Brazilian Congress of Mechanical Engineering*, vol I, Brasilia, Brasil, 181-184 (1993).

[79] Mavriplis, D.J., Adaptive mesh generation for viscous flows using delauney triangulation, *J. Comp. Phys.*, **90 (2)**, 271-291 (1990).

[80] Mavriplis, D.J., Unstructured mesh generation and adaptivity, ICASE report no.95-26, NASA CR195069, April 1995.

[81] Parsons, I.D., Parallel adaptive multigrid methods for elasticity, plasticity and eigenvalue problems, in: M. Papadrakakis, ed., *Solving Large-scale Problems in Mechanics - Parallel and Distributed Computer Applications*, vol. II, John Wiley & Sons Ltd, Chichester, England (1996).

[82] Bout, A., A displacement-based geometrically non-linear constant stress element, *Int. J. Numer. Methods Eng.*, **36**, 1161-1188 (1993).

[83] Durlofsky, L.J., Accuracy of mixed and control volume finite element approximations to Darcy velocity and related quatities, *Water Resour. Res.*, **30 (4)**, 965-973 (1994).

[84] Luo, H., Baum, J.D., Löhner, R. and Cabello, J., Adaptive edge-based finite element schemes for the Euler and Navier-Stokes equations on unstructured grids, *31st Aerospace Sciences meeting & Exhibit*, Reno, NV, USA, AIAA -93-0336. (1993).