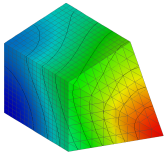


# Using LibMesh for Scientific Computations

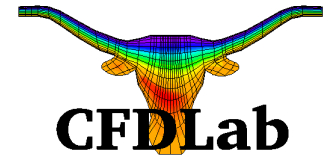
Roy Stogner  
John Peterson

<http://libmesh.sf.net>

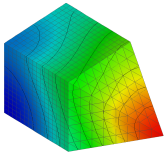
November 3, 2005



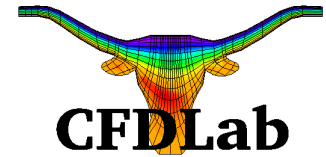
# Outline



- A Model Problem
- Galerkin FE Method
- Penalty Boundary Conditions
- Adaptivity
- Error Indicators
- 1D Example



# Model Problem



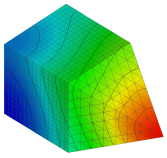
- Consider the 1D model ODE

$$\begin{cases} -u'' + bu' + cu = f & \in \Omega = (0, L) \\ u(0) = u_0 \\ u(L) = u_L \end{cases} \quad (1)$$

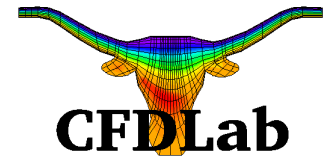
- with weak form

$$\int_{\Omega} (u'v' + bu'v + cuv) \, dx = \int_{\Omega} f v \, dx \quad (2)$$

for every  $v \in H_0^1(\Omega)$ .



## Model Problem (cont.)

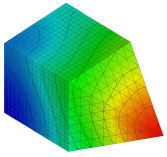


- The analogous  $d$ -dimensional problem with  $\Omega \subset \mathbb{R}^d$  and boundary  $\partial\Omega$  is

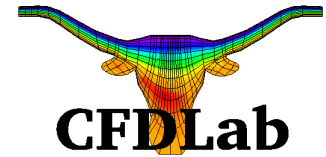
$$\begin{cases} -\Delta u + \mathbf{b} \cdot \nabla u + cu & = f & \in \Omega \\ u & = g & \in \partial\Omega \end{cases} \quad (3)$$

- with weak form

$$\int_{\Omega} (\nabla u \cdot \nabla v + (\mathbf{b} \cdot \nabla u)v + cuv) \, dx = \int_{\Omega} f v \, dx \quad (4)$$



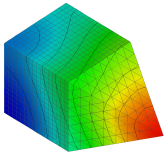
## Model Problem (cont.)



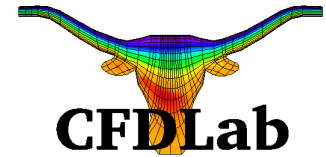
- The finite element method works with the weak form, replacing the trial and test functions  $u, v$  with their approximations  $u^h, v^h$ , and summing the contributions of the element integrals

$$\sum_{e=1}^{N_e} \int_{\Omega_e} (\nabla u^h \cdot \nabla v^h + (\mathbf{b} \cdot \nabla u^h) v^h + cu^h v^h - f v^h) dx = 0 \quad (5)$$

- Remark: We considered here a standard piecewise continuous finite element basis. In general,  $\nabla u^h$  will have a jump discontinuity across element boundaries.



# Galerkin FE Method



- Expressing  $u^h$  and  $v^h$  in our chosen piecewise continuous polynomial basis

$$u^h = \sum_{j=1}^N u_j \varphi_j \quad v^h = \sum_{i=1}^N c_i \varphi_i \quad (6)$$

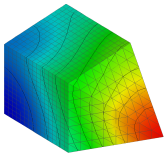
we obtain on each element  $\Omega_e$

$$\sum_{j=1}^N u_j \left[ \int_{\Omega_e} (\nabla \varphi_j \cdot \nabla \varphi_i + (\mathbf{b} \cdot \nabla \varphi_j) \varphi_i + c \varphi_j \varphi_i) dx \right] = \int_{\Omega_e} f \varphi_i dx \quad (7)$$

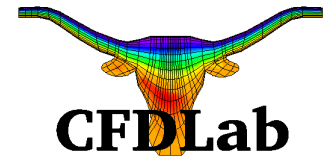
for  $i = 1 \dots N$ .

- In the standard element-stiffness matrix form,

$$\mathbf{K}_e \mathbf{U} = \mathbf{F}_e \quad (8)$$



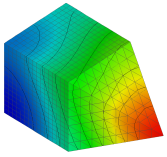
# LibMesh Representation



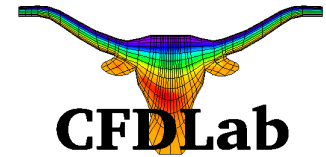
- To code the model problem in LibMesh, the user must provide the routine which computes  $\mathbf{K}_e$  and  $\mathbf{F}_e$  for each element.
- The integrals are computed over the reference element using an appropriate numerical quadrature rule with weights  $w_q$  and points  $\xi_q$ ,  $q = 1 \dots N_q$ .

$$\int_{\Omega_e} f \varphi_i dx = \int_{\hat{\Omega}_e} f \varphi_i |J| d\xi \approx \sum_{q=1}^{N_q} w_q |J(\xi_q)| f(\xi_q) \varphi_i(\xi_q) \quad (9)$$

- LibMesh provides the following variables for constructing  $\mathbf{K}_e$  and  $\mathbf{F}_e$  at quadrature point  $q$ :
  - $\text{JxW}[q]$  = the scalar value of the element Jacobian map times the quadrature rule weight
  - $\text{phi}[i][q] = \varphi_i(\xi_q)$
  - $\text{dphi}[i][q] = (J^{-1} \cdot \nabla_{\xi} \varphi_i)(\xi_q)$  (e.g. in 1D, this is  $\frac{\partial \phi_i}{\partial \xi} \frac{\partial \xi}{\partial x}(\xi_q)$ )



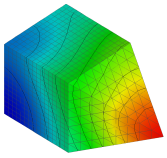
## LibMesh Representation (cont.)



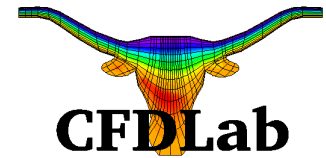
```
for (q=0; q<Nq; ++q) {
  // Compute b, c, f at this quadrature point
  // ...

  for (i=0; i<N; ++i) {
    Fe(i) += JxW[q]*f*phi[i][q];

    for (j=0; j<N; ++j)
      Ke(i,j) += JxW[q]*(
        (dphi[i][q]*dphi[j][q]) +
        (b*dphi[j][q])*phi[i][q] +
        c*phi[j][q]*phi[i][q]
      );
  }
}
```

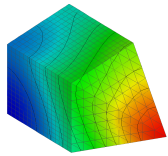


# Boundary Conditions

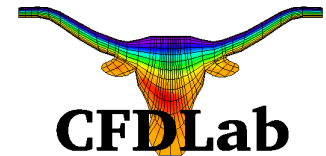


- Dirichlet boundary conditions are typically enforced after the global stiffness matrix  $\mathbf{K}$  has been assembled
- This usually involves
  1. placing a “1” on the main diagonal of the global stiffness matrix
  2. zeroing out the row entries
  3. placing the Dirichlet value in the rhs vector
  4. subtracting off the column entries from the rhs

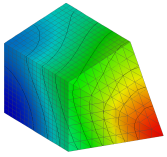
$$\begin{bmatrix} k_{11} & k_{12} & k_{13} & \cdot \\ k_{21} & k_{22} & k_{23} & \cdot \\ k_{31} & k_{32} & k_{33} & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix}, \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \cdot \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & k_{22} & k_{23} & \cdot \\ 0 & k_{32} & k_{33} & \cdot \\ 0 & \cdot & \cdot & \cdot \end{bmatrix}, \begin{bmatrix} g_1 \\ f_2 - k_{21}g_1 \\ f_3 - k_{31}g_1 \\ \cdot \end{bmatrix}$$



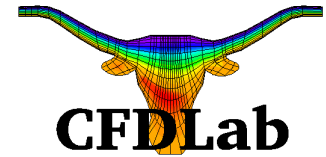
## Boundary Conditions (cont.)



- This approach works for an interpolary finite element basis but not in the general case.
- For large problems with parallel sparse matrices, it is inefficient to change individual entries once the global matrix is assembled.
- What is required is a way to enforce boundary conditions for a generic finite element basis *at the element stiffness matrix level*.
- Solution: “Penalty” Boundary Conditions



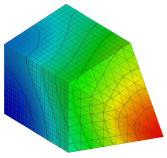
# Penalty Boundary Conditions



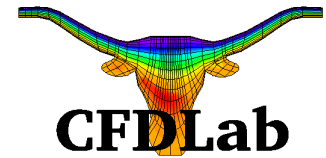
- An additional “penalty” term is added to the standard weak form

$$\int_{\Omega} (\nabla u \cdot \nabla v + (\mathbf{b} \cdot \nabla u)v + cuv) dx + \underbrace{\frac{1}{\epsilon} \int_{\partial\Omega} (u - g)v dx}_{\text{penalty term}} = \int_{\Omega} f v dx$$

- Here  $\epsilon \ll 1$  is chosen so that, in floating point arithmetic,  $\frac{1}{\epsilon} + 1 = \frac{1}{\epsilon}$ .
- This weakly enforces  $u = g$  on the boundary at the element level, and works for general finite element bases.
- It requires a few additional calculations (edge/face integrals) but is more efficient than modifying row entries after assembly



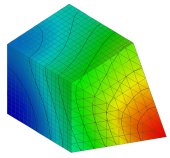
# LibMesh Representation



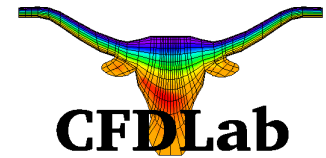
LibMesh provides:

- A quadrature rule with  $N_{qf}$  points and  $JxW\_f []$
- A finite element coincident with the boundary face that has  $N_f$  shape function values  $\phi\_f [] []$

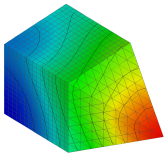
```
for (qf=0; qf<Nqf; ++qf) {  
    // Compute g at this face quadrature point  
  
    for (i=0; i<Nf; i++) {  
        Fe(i) += JxW_f[qf]*penalty*g*phi_face[i][qf];  
  
        for (j=0; j<Nf; j++)  
            Ke(i,j) += JxW_f[qf]*penalty*phi_f[i][qf]*phi_f[j][qf];  
        }  
    }
```



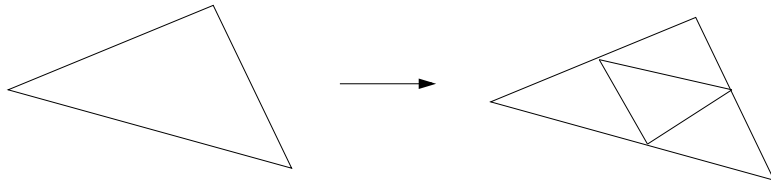
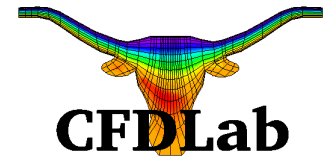
# Adaptivity And Error Indicators



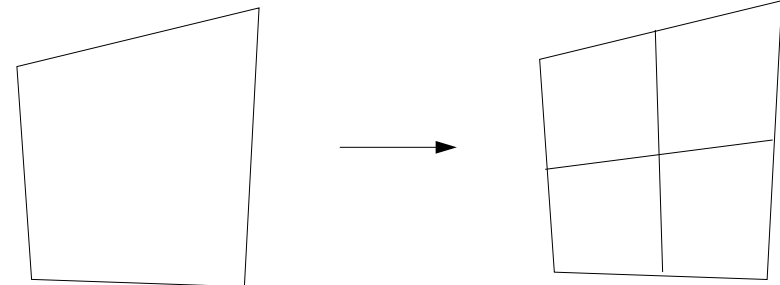
- A major goal of the LibMesh library is to provide:
  - Adaptive mesh refinement support for standard geometric elements
  - Generic, physics-independent error indicators
- In this context, we'll discuss
  - “Natural” refinement patterns
  - A flux-jump error indicator



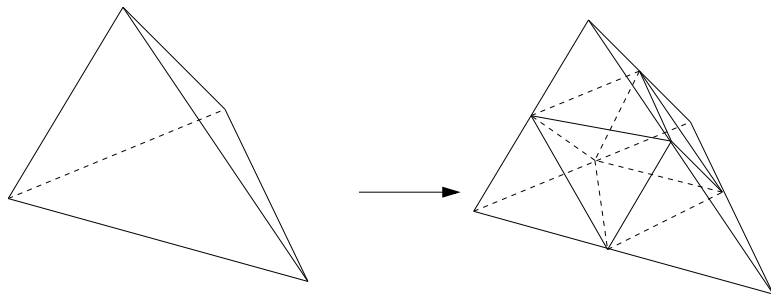
# Natural Refinement Patterns



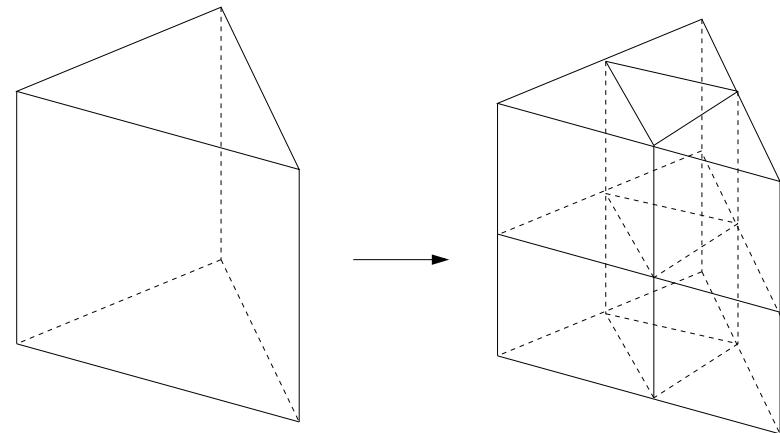
Triangle



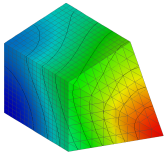
Quadrilateral



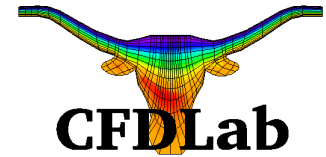
Tetrahedron



Prism



# Flux-Jump Error Indicator



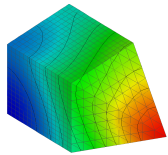
- The flux-jump error indicator is derived starting from the element integrals

$$\sum_{e=1}^{N_e} \int_{\Omega_e} (\nabla u^h \cdot \nabla v^h + (\mathbf{b} \cdot \nabla u^h) v^h + c u^h v^h - f v^h) dx = 0 \quad (5)$$

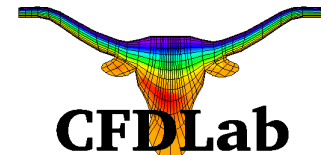
- Applying the divergence theorem “in reverse” obtains

$$\sum_{e=1}^{N_e} \int_{\Omega_e} (-\Delta u^h + (\mathbf{b} \cdot \nabla u^h) + c u^h - f) v^h dx + \quad (10)$$

$$\sum_{\partial\Omega_e \not\subset \partial\Omega} \int_{\partial\Omega_e} \left[ \frac{\partial u^h}{\partial n} \right] v^h dx = 0$$



## Flux-Jump Error Indicator (cont.)



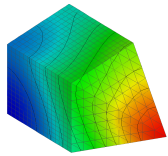
- Defining the cell residual

$$r(u^h) = -\Delta u^h + (\mathbf{b} \cdot \nabla u^h) + cu^h - f \quad (11)$$

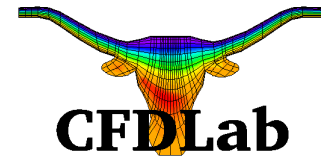
we have

$$\sum_{e=1}^{N_e} \int_{\Omega_e} r(u^h) v^h dx + \sum_{\partial\Omega_e \not\subset \partial\Omega} \int_{\partial\Omega_e} \left[ \frac{\partial u^h}{\partial n} \right] v^h dx = 0 \quad (12)$$

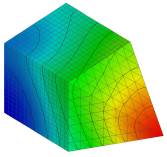
- Clearly, the exact solution  $u$  satisfies (12) identically.
- Computing  $r(u^h)$  requires knowledge of the differential operator (i.e. knowledge of the “physics”).
- The second sum leads to a *physics-independent* method for estimating the error in the approximate solution  $u^h$ .



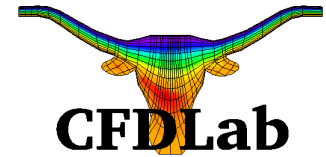
## Flux-Jump Error Indicator (cont.)



- Pros
  - Ideal for low-order (piecewise linear) elements
  - Easily extensible to adaptivity with hanging nodes
  - Works well in practice for nonlinear, time-dependent problems, and problems with shocks, layers, discontinuities, etc.
- Cons
  - For higher-order elements, the interior residual term may dominate
  - Relatively expensive to compute
  - Makes no sense for discontinuous and  $C^1$  FE bases



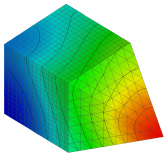
# 1D Example



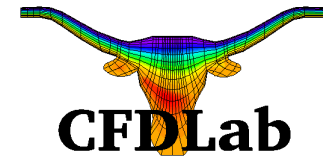
- In 1 dimension, the jump integrals reduce to point-wise evaluation of the derivatives at the element boundaries.
- For linear elements, the error indicator  $\eta$  for a particular element  $\Omega_e = (x_e, x_{e+1})$  is defined as

$$\eta^2 = \frac{h_e}{N_{\text{int}}} \sum_{i=1}^{N_{\text{int}}} \llbracket u'(y_i) \rrbracket^2 \quad (13)$$

where  $h_e = x_{e+1} - x_e$  is the element length, and  $N_{\text{int}} \leq 2$  is the number of *interior* nodes  $y_i$  the element has.



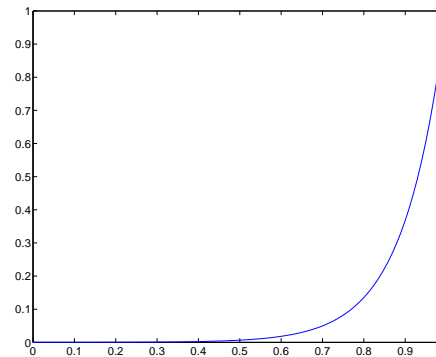
## 1D Example (cont.)



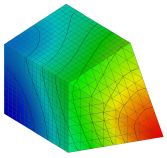
- Consider the function

$$u = \frac{1 - \exp(10x)}{1 - \exp(10)} \quad (14)$$

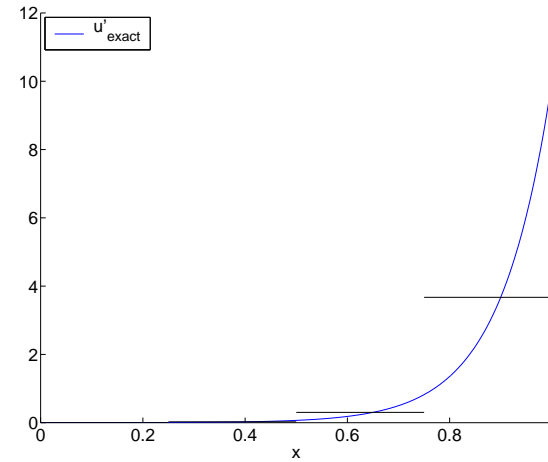
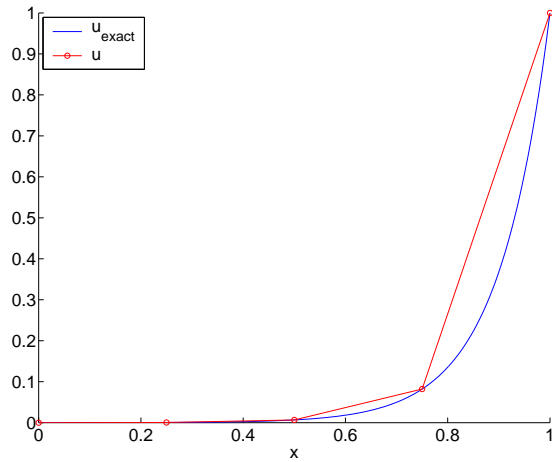
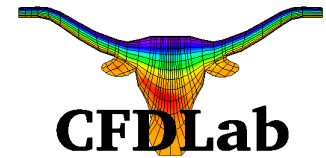
which is a solution of the classic 1D advection-diffusion boundary layer equation.



- We assume here that the finite element solution is the linear interpolant of  $u$ , and compute the error indicator for a sequence of uniformly refined grids.

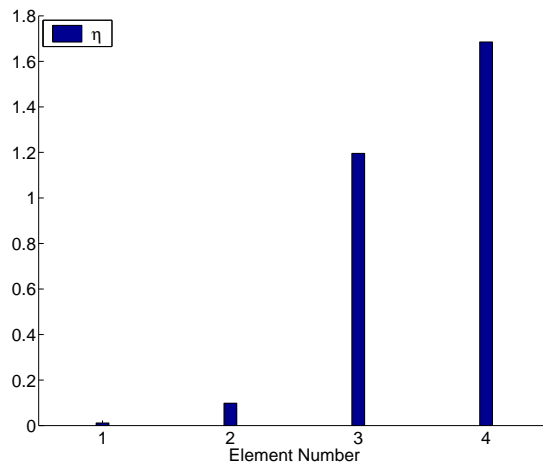


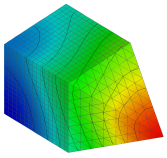
# 1D Example (cont.)



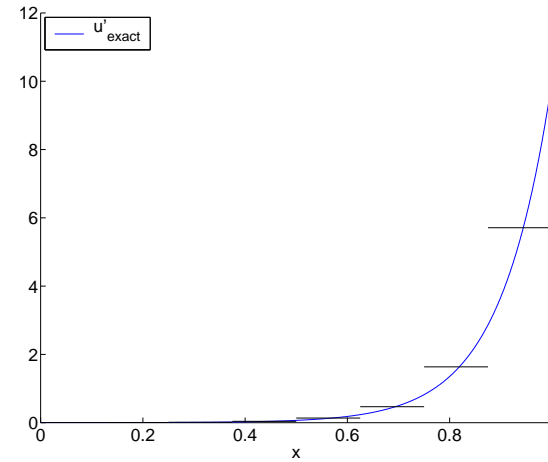
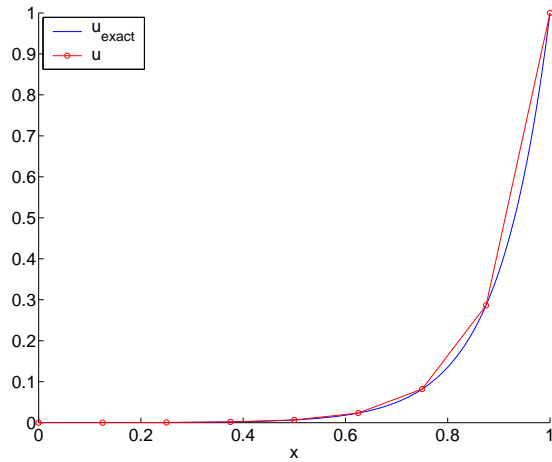
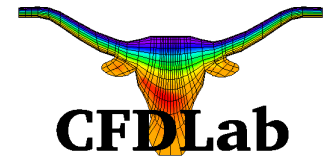
4 elements

$$\|e\|_{L_2} = 0.09$$



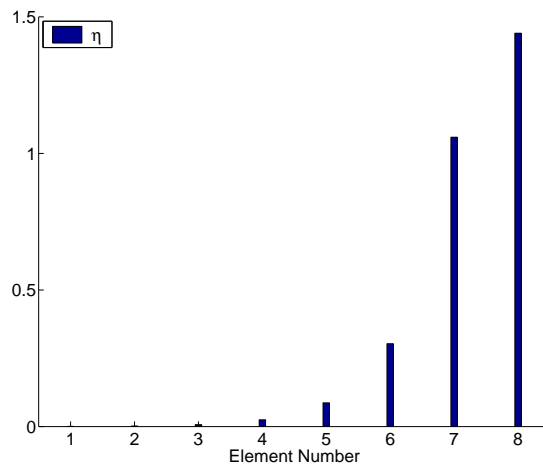


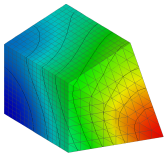
# 1D Example (cont.)



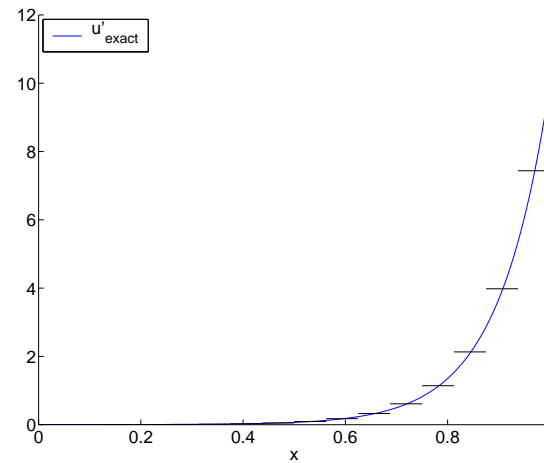
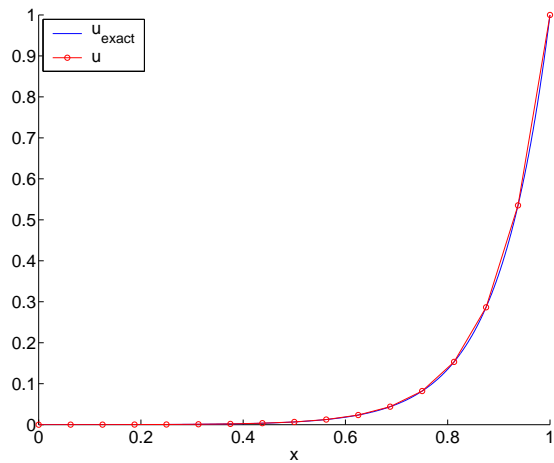
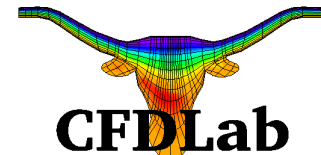
8 elements

$$\|e\|_{L_2} = 0.027$$



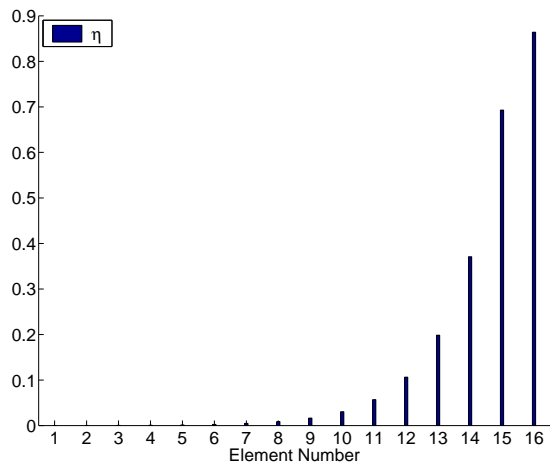


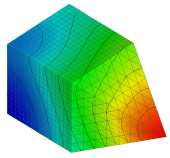
# 1D Example (cont.)



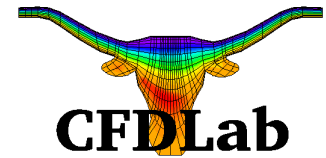
16 elements

$$\|e\|_{L_2} = 0.0071$$



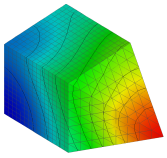


# A Simple Refinement Strategy

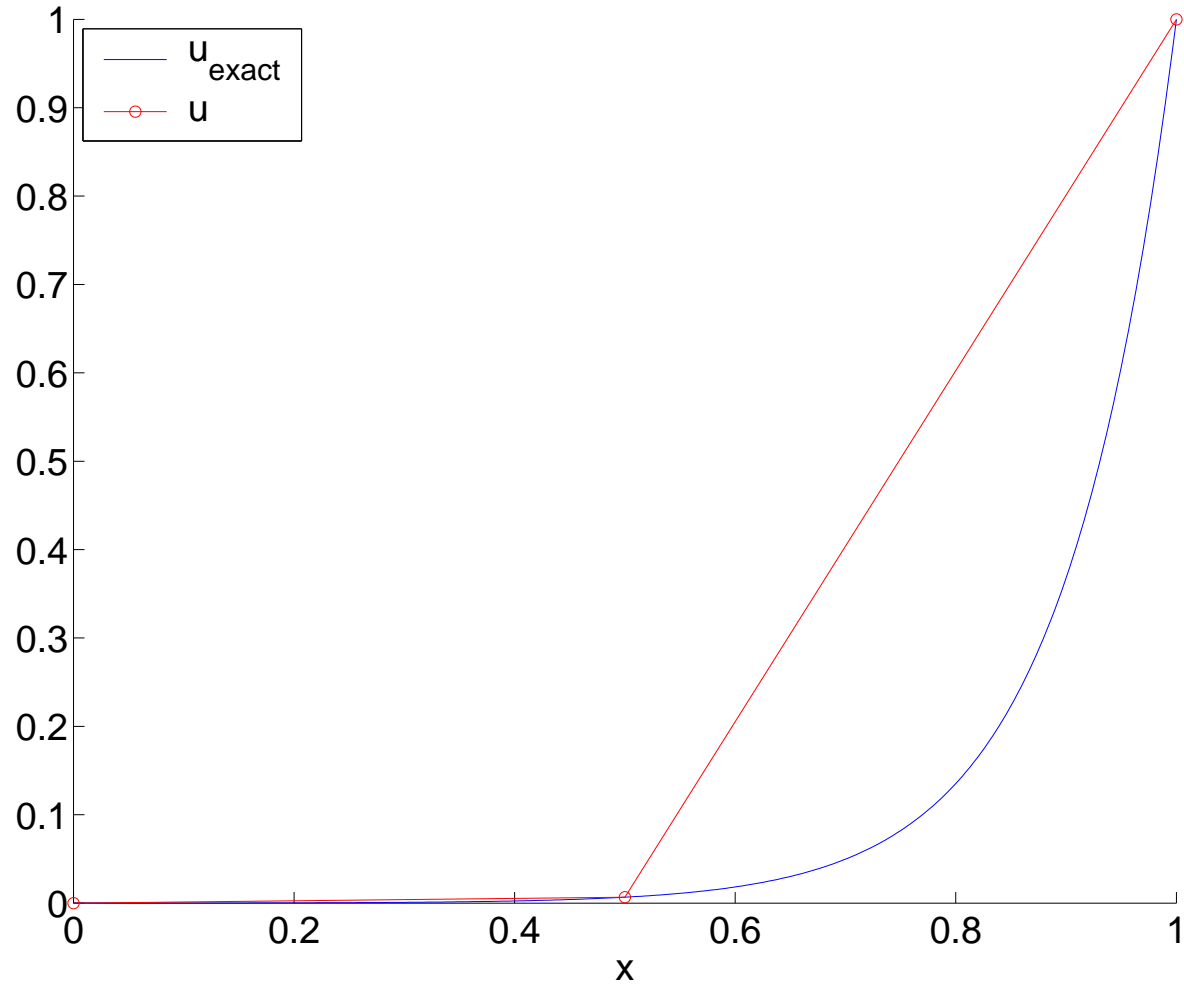
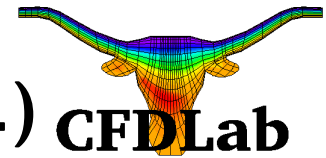


- A simple adaptive refinement strategy with  $r_{\max}$  refinement steps for this 1D example problem is:

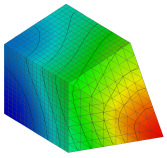
```
r=0;
while (r < r_max)
  Compute the FE solution (linear interpolant)
  Estimate the error (using flux-jump indicator)
  Refine the elements whose error is in the top 10%
  Increment r
end
```



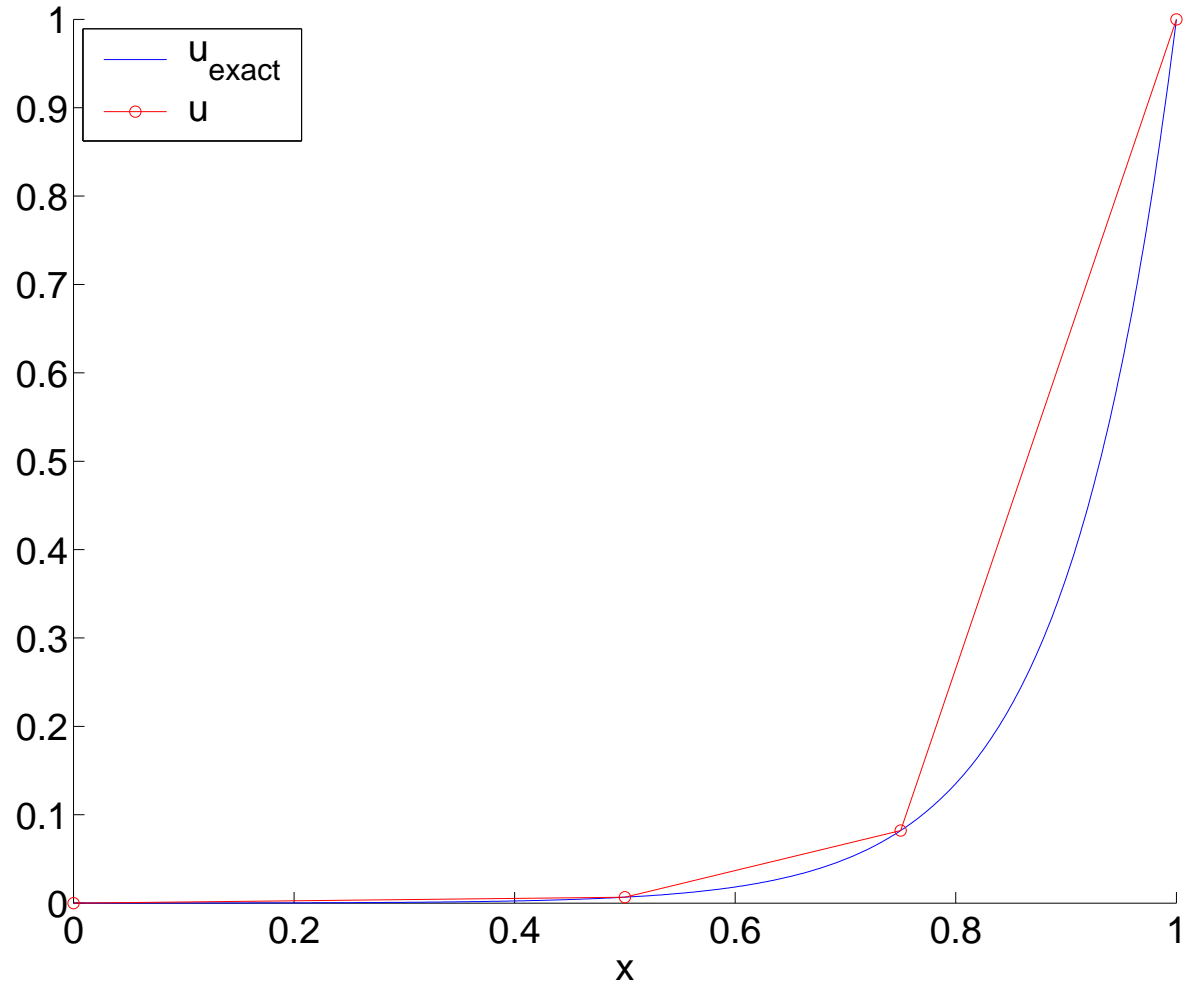
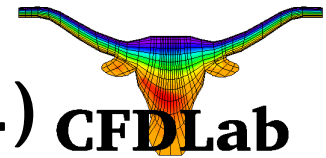
# A Simple Refinement Strategy (cont.) **CFDLab**



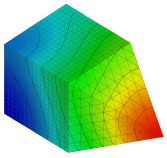
2 elements



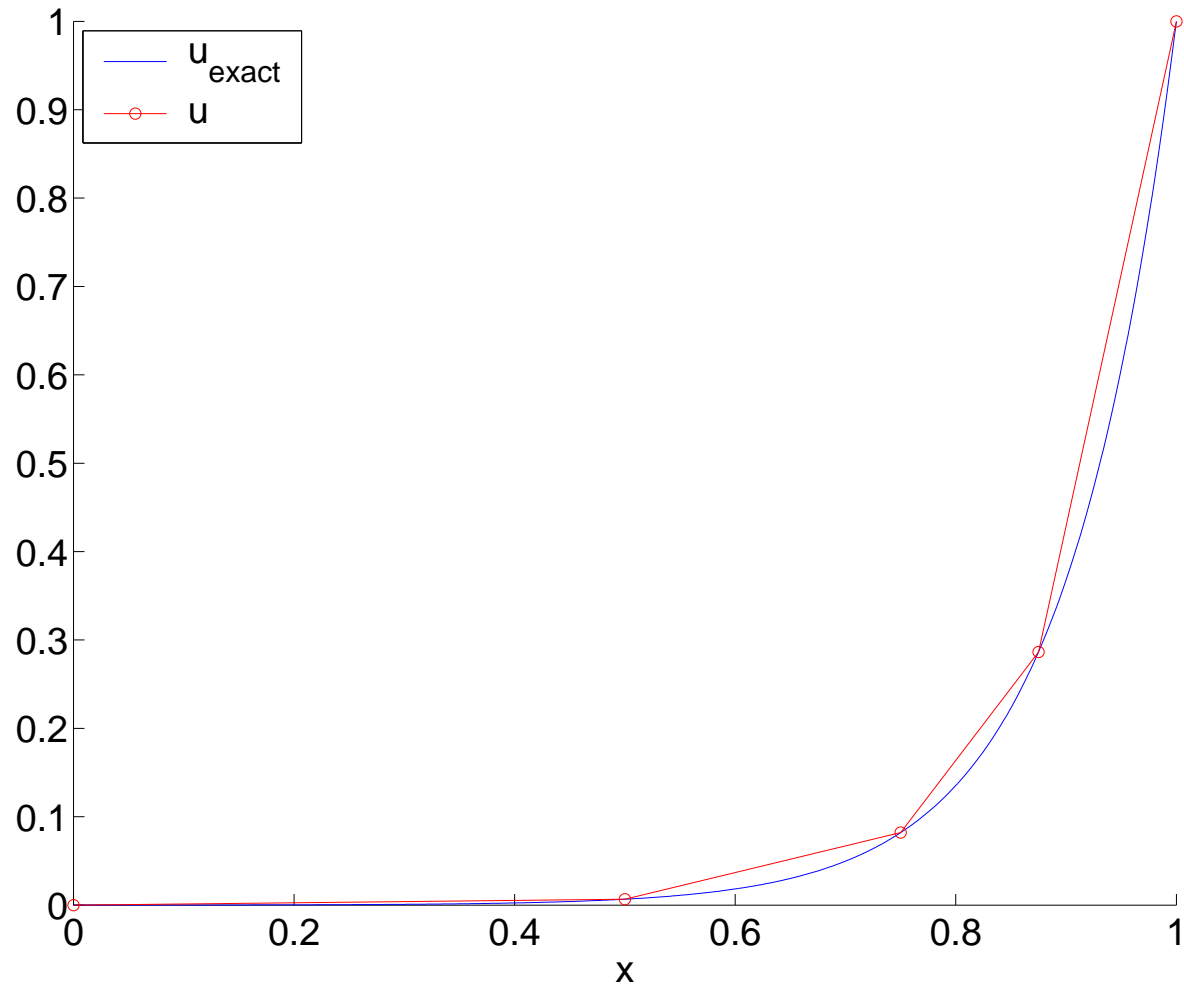
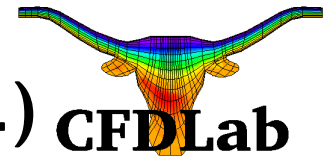
# A Simple Refinement Strategy (cont.)



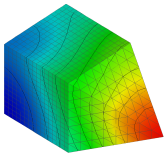
3 elements



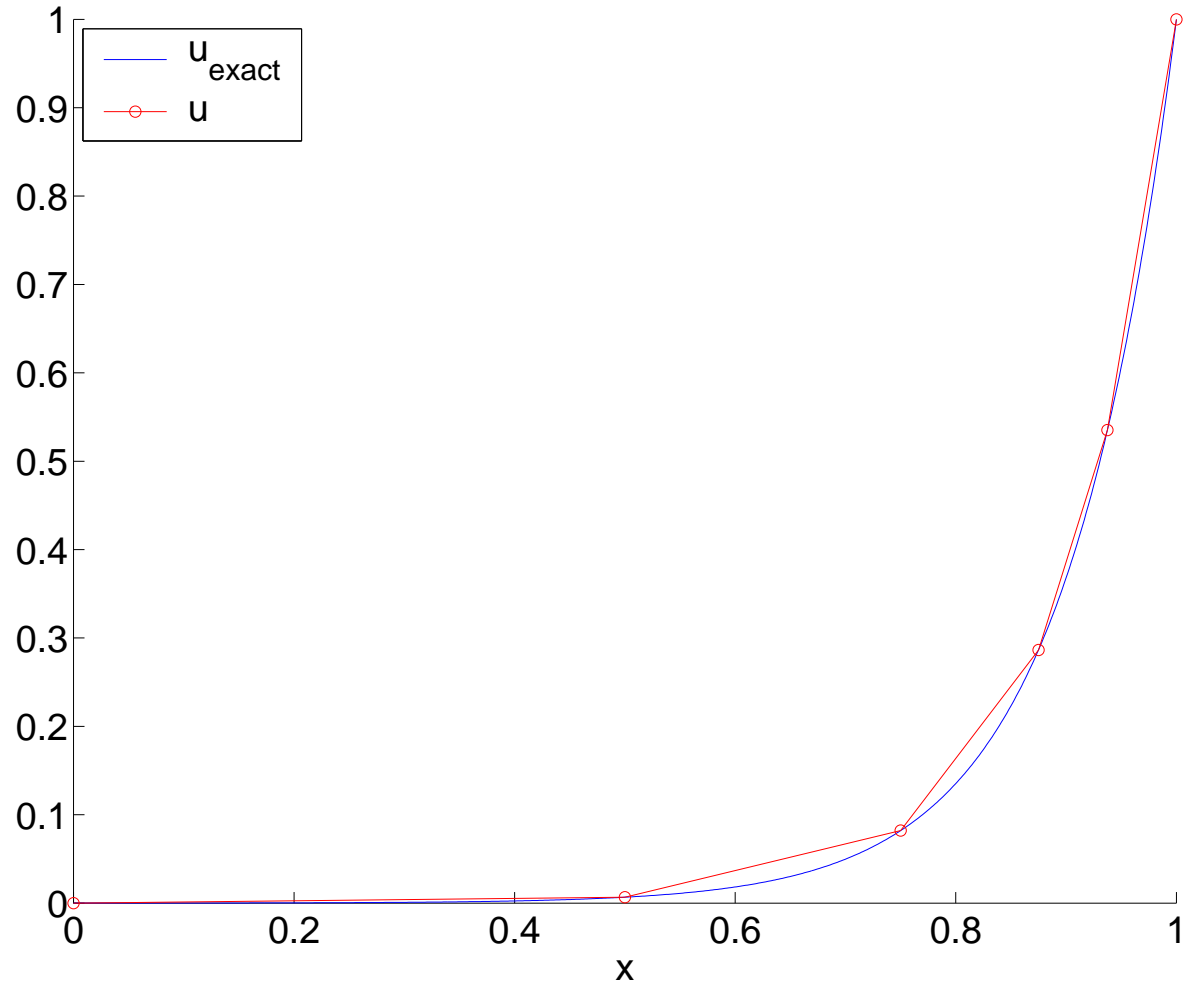
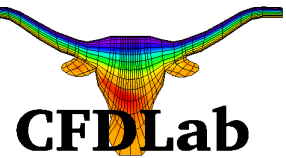
# A Simple Refinement Strategy (cont.)



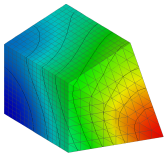
4 elements



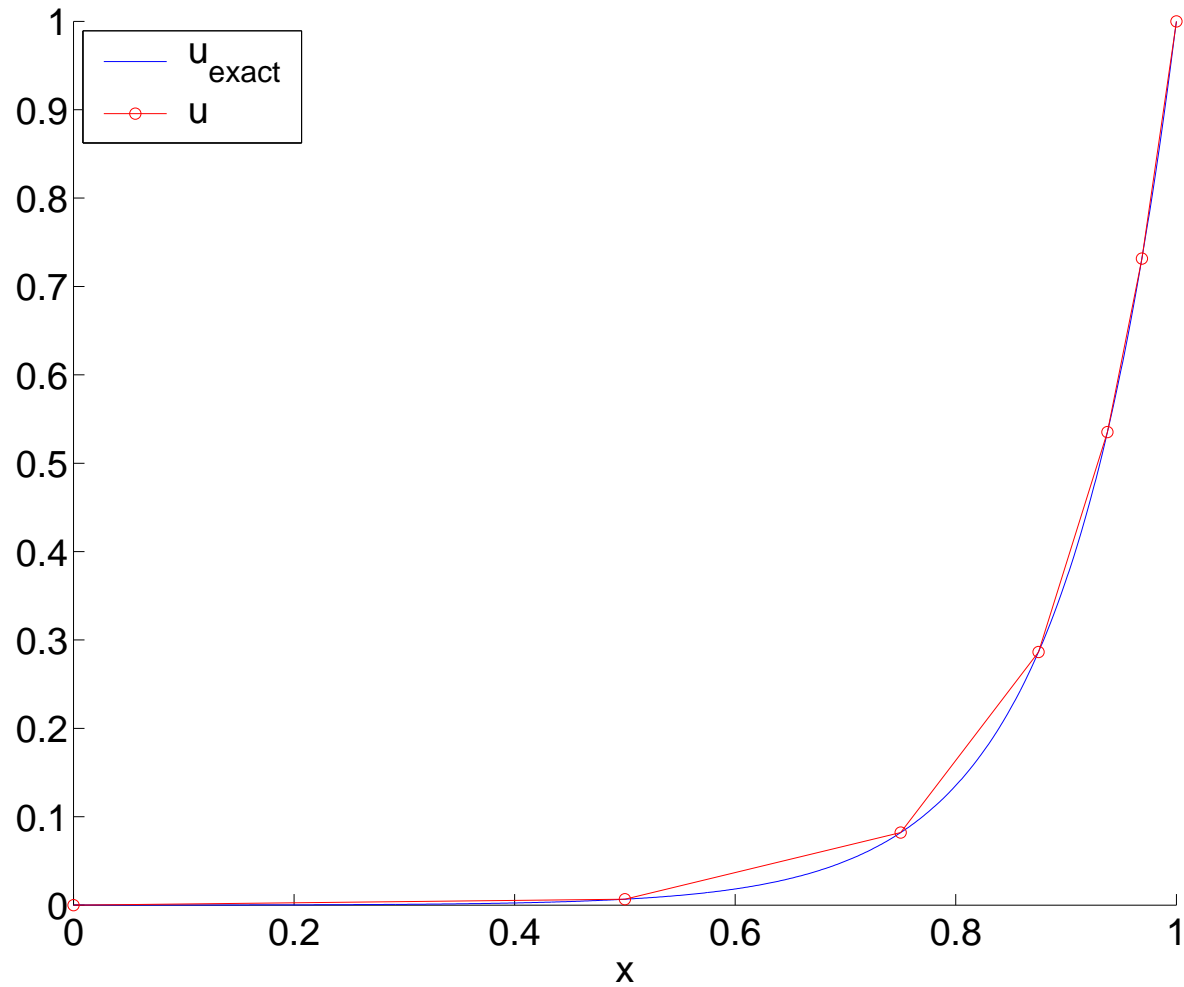
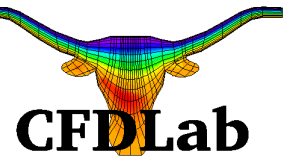
# A Simple Refinement Strategy (cont.)



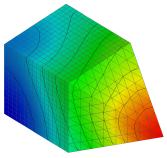
5 elements



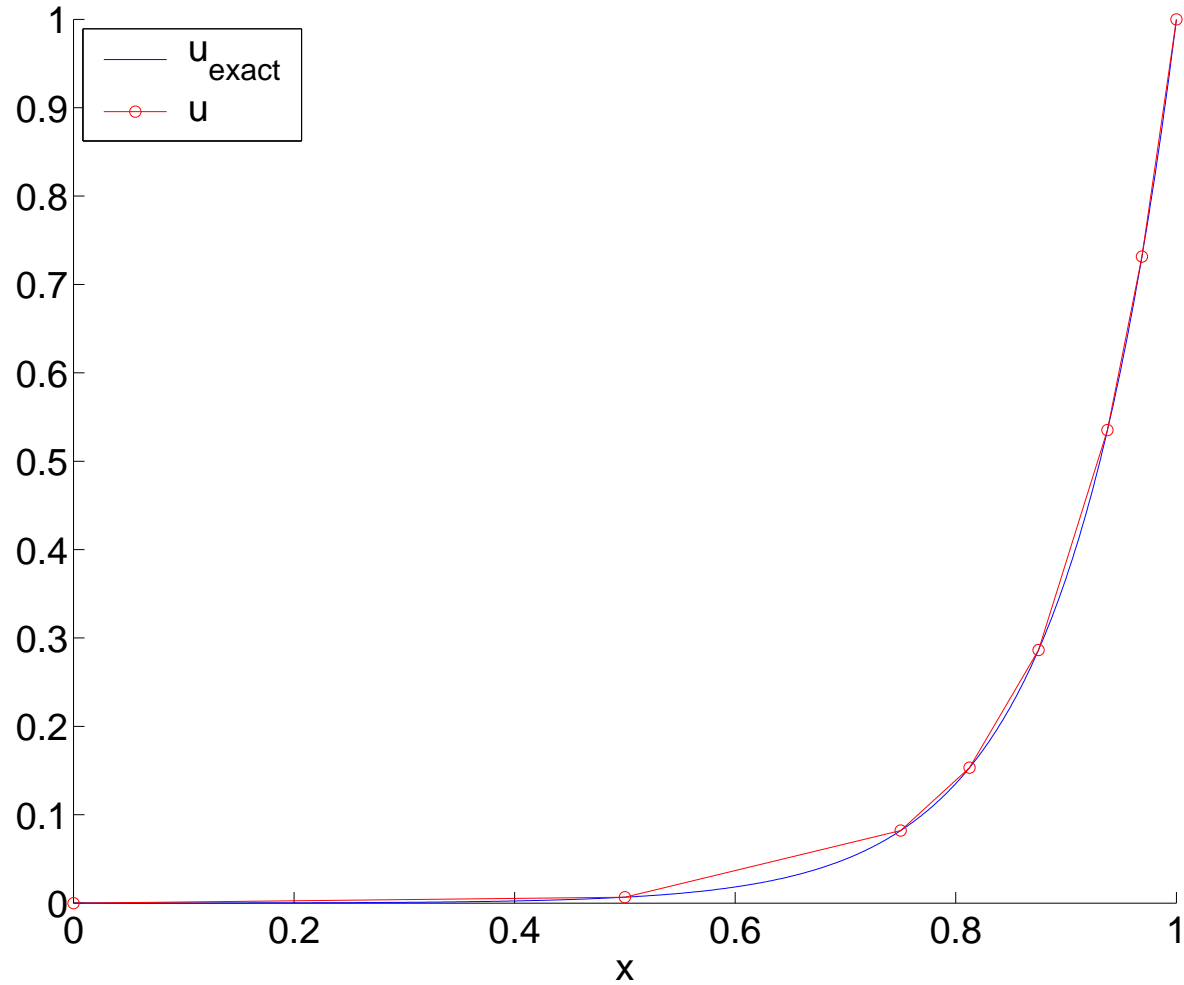
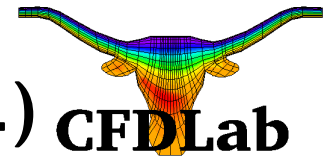
# A Simple Refinement Strategy (cont.)



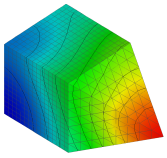
6 elements



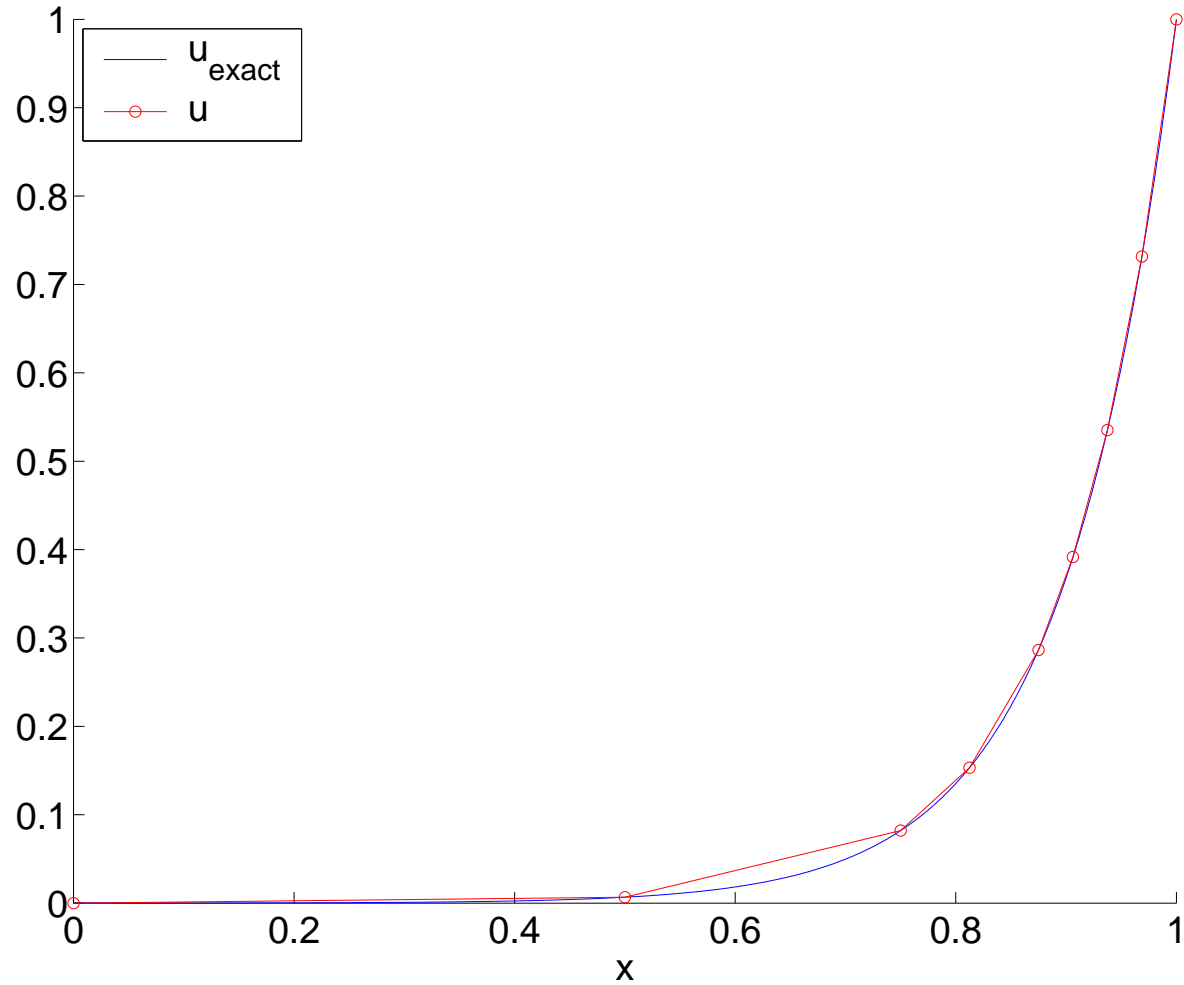
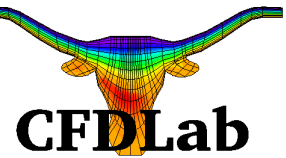
# A Simple Refinement Strategy (cont.)



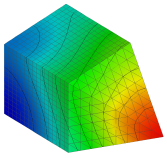
7 elements



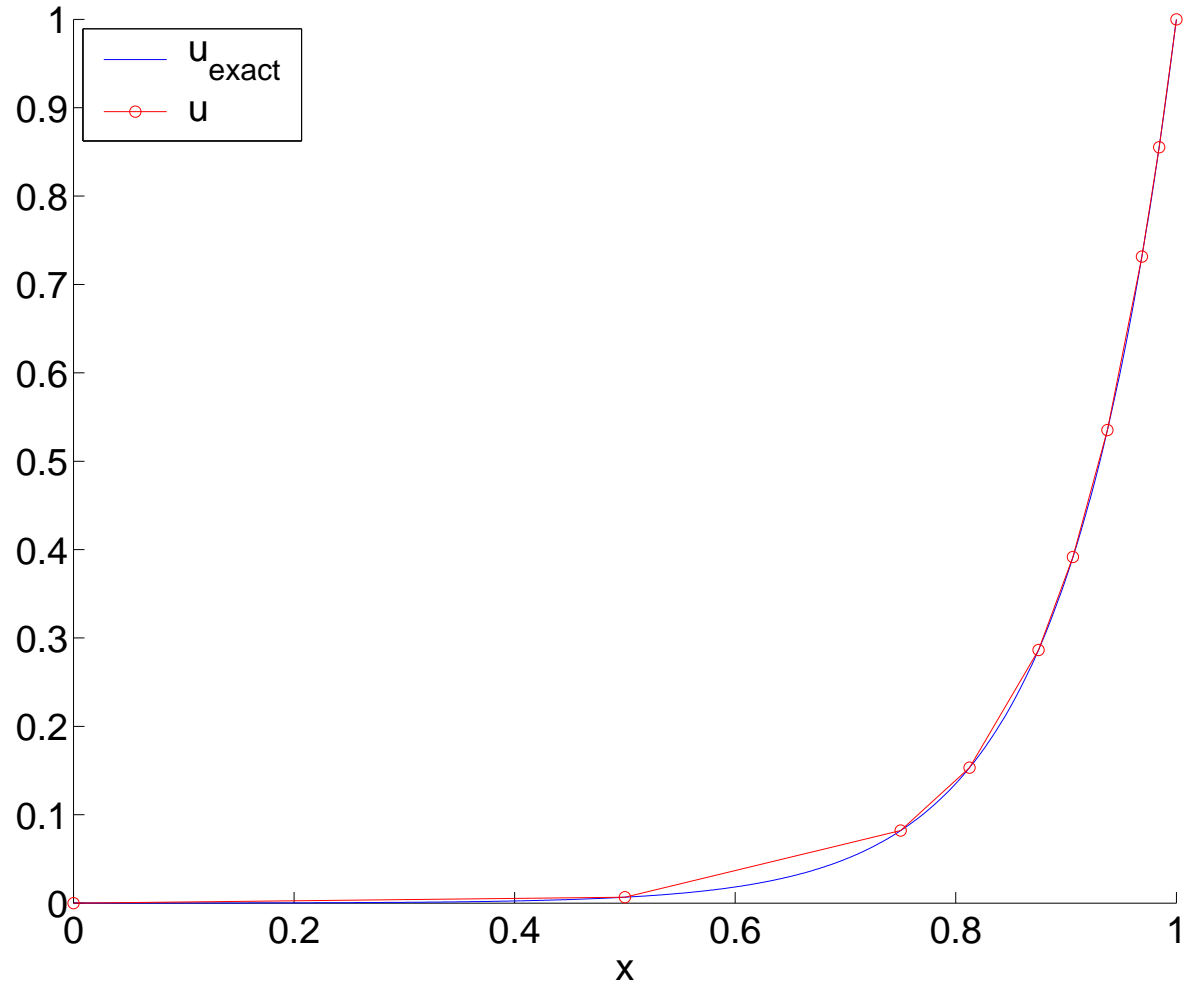
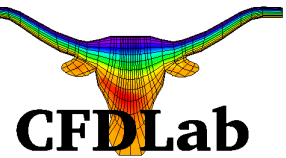
# A Simple Refinement Strategy (cont.)



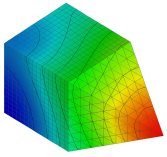
8 elements



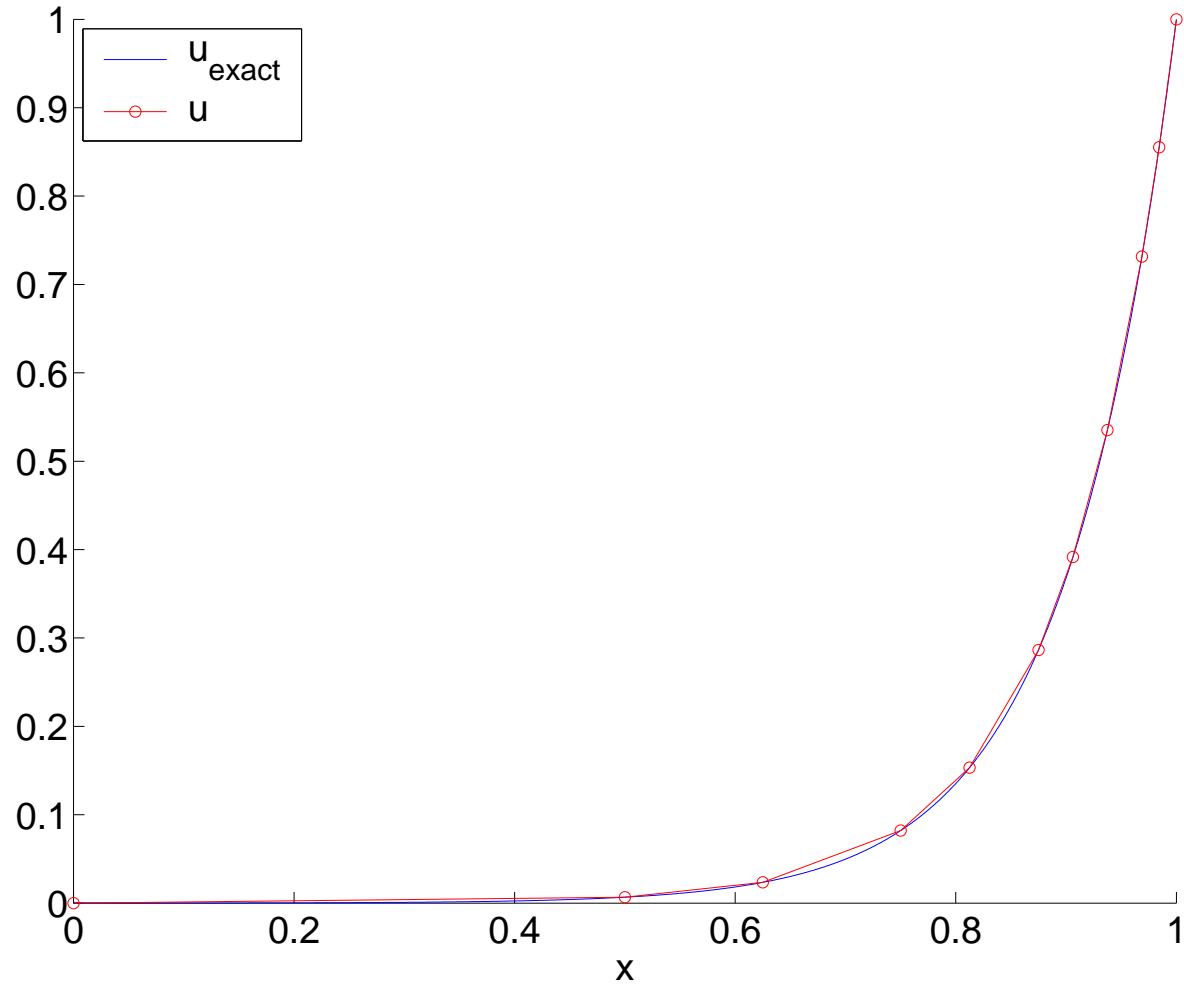
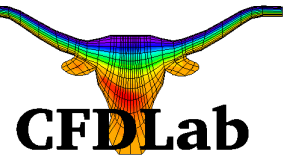
# A Simple Refinement Strategy (cont.)



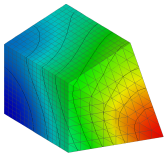
9 elements



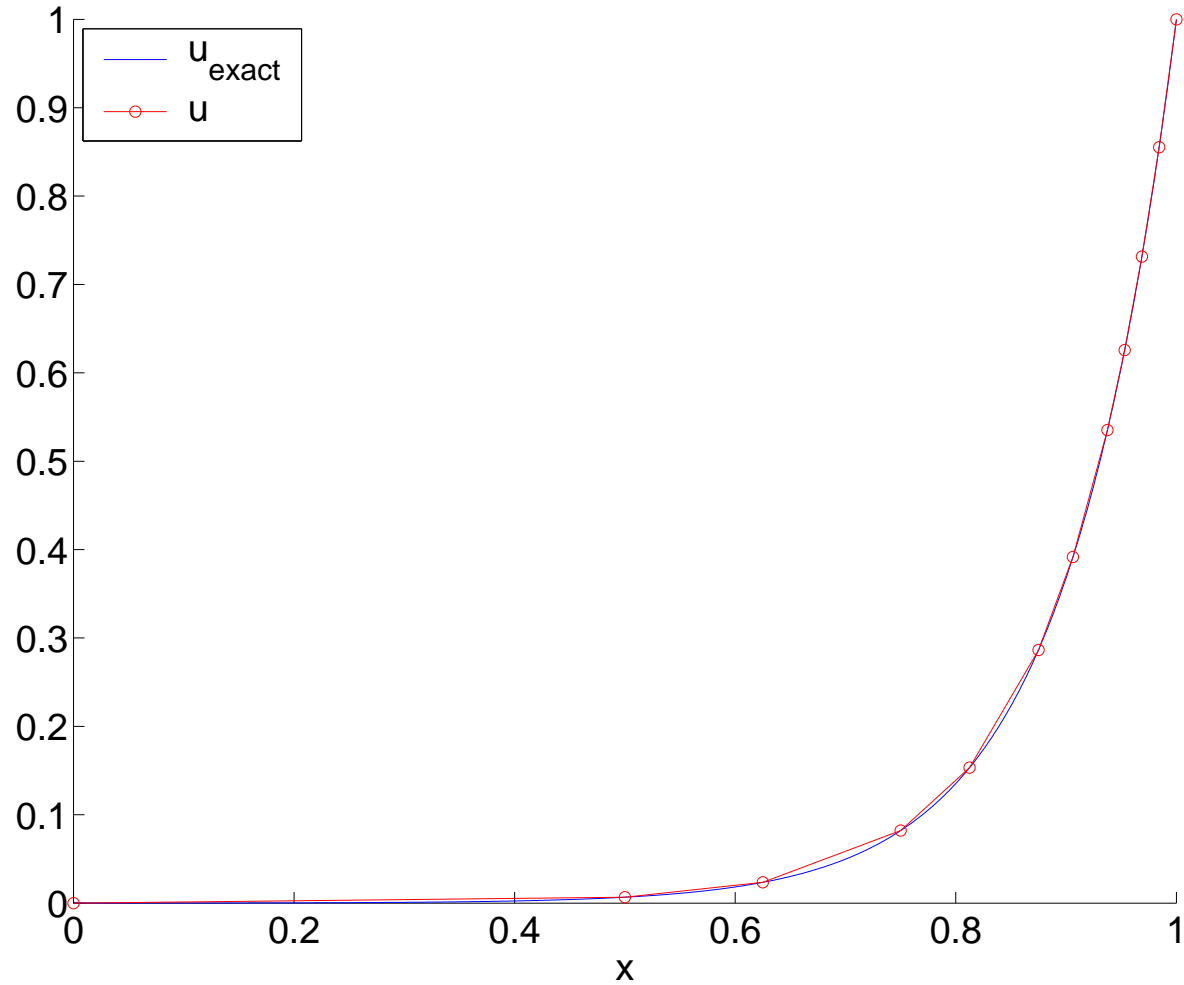
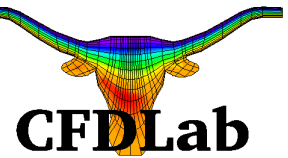
# A Simple Refinement Strategy (cont.)



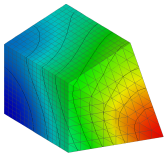
10 elements



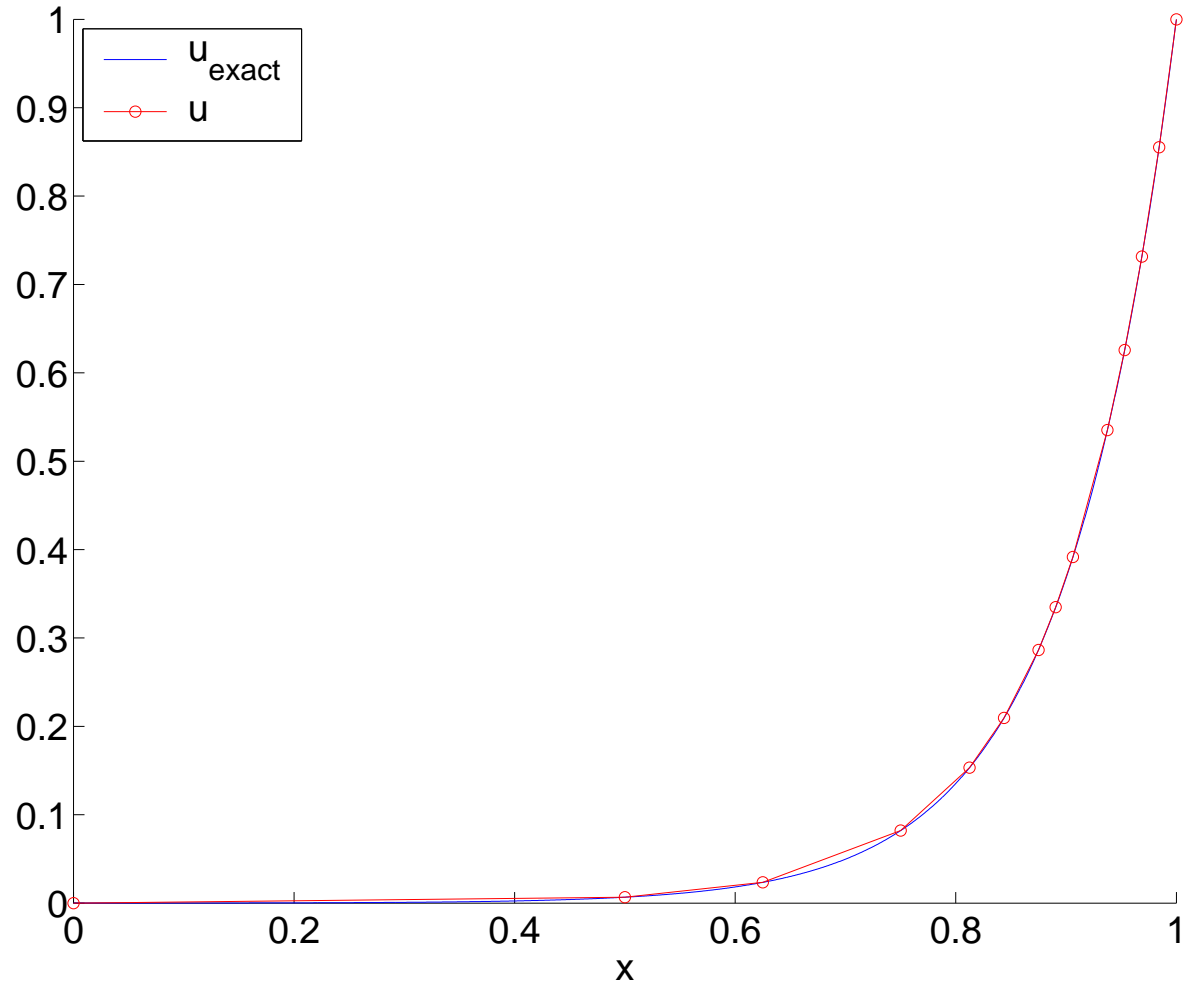
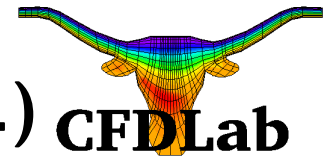
# A Simple Refinement Strategy (cont.)



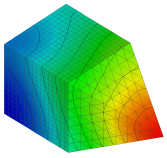
11 elements



# A Simple Refinement Strategy (cont.)



Final: 13 elements



# A Simple Refinement Strategy (cont.) **CFDLab**

