CS 3200
Introduction to Scientific Computing

Instructor: Martin Berzins

Topic: Adaptive Methods

Motivation

- Not only do we want to compute the correct solution but we would like do so efficiently
- When implementing methods we would like them to automatically control the error
- Examples are linear interpolation and quadrature
- In both cases the error estimate depends on?

Motivation

- When implementing methods we would like them to automatically control the error
- Examples are linear interpolation and quadrature
- In both cases the error estimate depends on?
- The step size and the second derivative of the function $\Delta x^2 \ \ and\ f"(\zeta)$

Estimating Linear Interpolation Error

$$f(x+h) = f(x) + h\frac{df}{dx} + \frac{h^2}{2}\frac{d^2 f}{dx^2} + \frac{h^3}{6}\frac{d^3 f}{dx^3} + O(h^4)$$

$$f(x-h) = f(x) - h\frac{df}{dx} + \frac{h^2}{2}\frac{d^2 f}{dx^2} - \frac{h^3}{6}\frac{d^3 f}{dx^3} + O(h^4)$$

Adding these equations and subtracting 2*f*(*x*) gives

$$f(x+h) + f(x-h) - 2f(x) = h^2\frac{d^2 f}{dx^2} + O(h^4)$$

$$\frac{d^2 f}{dx^2} = \frac{f(x+h) + f(x-h) - 2f(x)}{h^2} + O(h^2)$$

For Example Interpolation

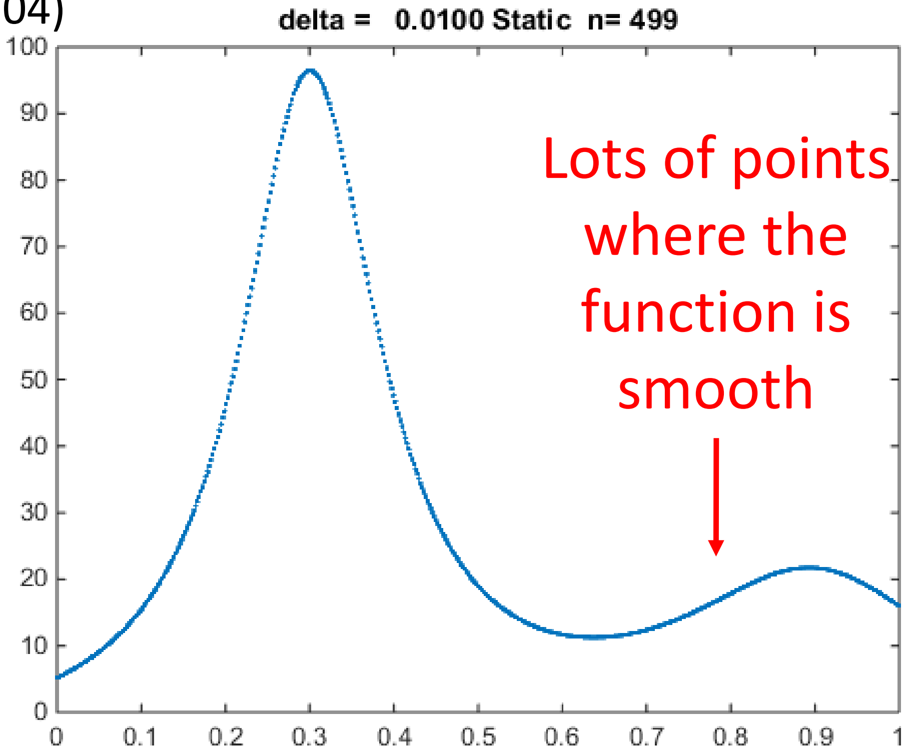
- Error over an interval bounded by* $M_2 \frac{h_{\max}^2}{8}$ maximum second derivative x maximum stepsize
- If we require this error to be less than a user specified error *tol* then number of points *n* defined by

$$h_{\max}^2 \le \frac{8tol}{M_2}, h_{\max} = \frac{(b-a)}{(n-1)}$$

$$n \ge 1 + (b-a)\sqrt{M_2 / 8}$$

Fixed linear Interpolation Applied to Matlab humps function

```
%   humps(x) = 1/((x-.3)^2 + .01)  +  1/((x-.9)^2+.04)
 close all
% Second derivative estimate based
% on divided differences
 z = linspace(0,1,101);
 humpvals = humps(z);
 M2 = max(abs(diff(humpvals,2)/(.01)^2));
 delta = 0.01
 figure
 [x,y] = pwLStatic('humps',M2,0,1,delta);
 plot(x,y,'.');
 title(sprintf('delta = %8.4f Static  n= %2.0f',delta,length(x)))
```



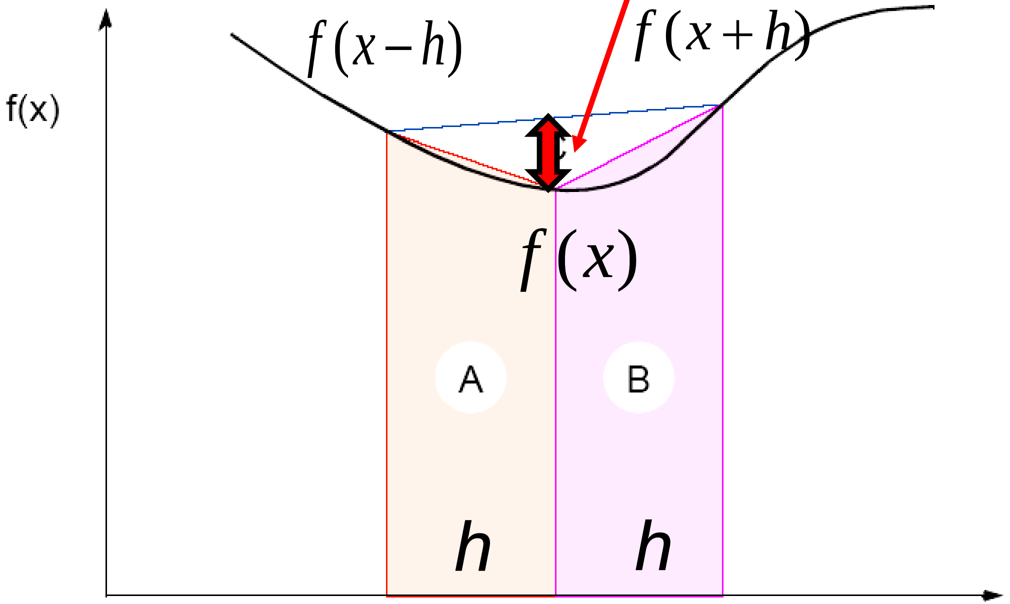
Adaptive Interpolation

Solution adapts to shape of curve. Use two approximations

$$Error = \left(\frac{f(x+h)+f(x-h)}{2} - f(x)\right)$$

$$\approx \frac{h^2}{2}\frac{d^2 f}{dx^2}$$

Error Estimate



Implementation MATLAB pwLAdapt

An interval if accepted if $$Error = \left(\frac{f(x+h)+f(x-h)}{2} - f(x)\right) < tol$$

Or if *h*< *hmin* *where tol is a user-defined tolerance*

Intervals are recursively defined until the test is passed or the minimum step hmin reached

This is implemented in the Matlab file   pwLAdapt.m

Please see the Canvas webpage

pwLAdapt(xL,fL,xR,fR,delta,hmin)

```
if (xR-xL) <= hmin
    % Subinterval is acceptable form mesh
  else
    mid  = (xL+xR)/2;
    fmid = f(mid);
    if (abs(((fL+fR)/2) - fmid) <= delta )
      % Subinterval accepted form mesh
    else
      % Produce left and right partitions, then synthesize.
      [xLeft,yLeft]   = pwLAdapt(xL,fL,mid,fmid,delta,hmin);
      [xRight,yRight] = pwLAdapt(mid,fmid,xR,fR,delta,hmin);
%         form mesh
      end
  end
```

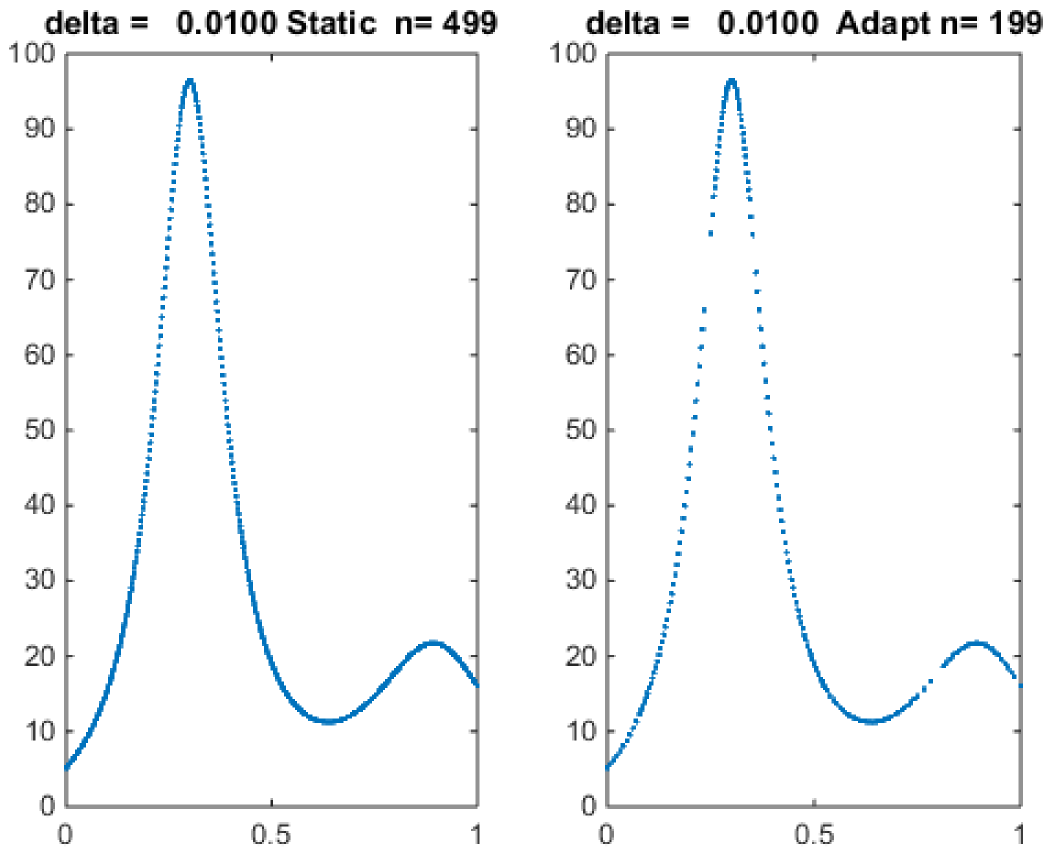
Note there is no mesh formation here just the logic, see full code for the rest

Recursive calls

```
% Script File: ShowPWL2    Compares pwLStatic and pwLAdapt on [0,1] using the function
%   humps(x) = 1/((x-.3)^2 + .01)  +  1/((x-.9)^2+.04)
  close all
z = linspace(0,1,101);
  humpvals = humps(z);
  M2 = max(abs(diff(humpvals,2)/(.01)^2));
  delta = 0.01
    figure
    [x,y] = pwLStatic('humps',M2,0,1,delta);
    subplot(1,2,1)
    plot(x,y,'.');
    title(sprintf('delta = %8.4f Static  n= %2.0f',delta,length(x)))
    [x,y] = pwLAdapt('humps',0,humps(0),1,humps(1),delta,.001);
    subplot(1,2,2)
    plot(x,y,'.');
    title(sprintf('delta = %8.4f  Adapt n= %2.0f',delta,length(x)))
```

Comparison of Adaptive vs Static

The adaptive approach uses only half the points



Richardson Extrapolation – an easy way to estimate the error

Let $I(trap,h)$ be the approximate value of the integral with the Trapezoidal rule and step h. Let $Iexact$ be the exact value.

Then $$Error(h) = Iexact - I(trap,h) \approx \frac{-h^2}{12} f''(\zeta_1)$$

and $$Error(\frac{h}{2}) = Iexact - I(trap,\frac{h}{2}) \approx \frac{-h^2/4}{12} f''(\zeta_1)$$

We don't know these but we know these

Extrapolation – an easy way to estimate the error

Subtract the bottom equation from the top one

$$Iexact - I(trap, h) \approx \frac{(-h^2)}{12} f''(\zeta_1)$$

- -

$$Iexact - I(trap, \frac{h}{2}) \approx \frac{1}{4}\frac{(-h^2)}{12} f''(\zeta_1)$$

= =

$$-I(trap, h) + I(trap, \frac{h}{2}) \approx \frac{3}{4}\frac{(-h^2)}{12} f''(\zeta_1)$$

Extrapolation – an easy way to estimate the error

Hence

$$\frac{3}{4}\frac{(-h^2)}{12} f''(\zeta_1) \approx -I(trap, h) + I(trap, \frac{h}{2})$$

or

$$\frac{4}{3}(-I(trap, h) + I(trap, \frac{h}{2})) \approx \frac{(-h^2)}{12} f''(\zeta_1)$$

or

$$Error(h) \approx \frac{4}{3}(-I(trap, h) + I(trap, \frac{h}{2}))$$

Extrapolation – an easy way to estimate the error

As

$$Error(h) \approx \frac{4}{3}(-I(trap,h) + I(trap,\frac{h}{2}))$$

and

$$Error(\frac{h}{2}) \approx \frac{1}{4} Error(h)$$

then

$$Error(\frac{h}{2}) = \frac{1}{3}(-I(trap,h) + I(trap,\frac{h}{2}))$$

Extrapolation – an easy way to estimate the error

As

$$Error(h) \approx \frac{4}{3}(-I(trap,h) + I(trap,\frac{h}{2}))$$

and

$$Error(\frac{h}{2}) \approx \frac{1}{4}Error(h)$$

then

$$Error(\frac{h}{2}) = \frac{1}{3}(-I(trap,h) + I(trap,\frac{h}{2}))$$

In other words by repeating the calculation twice with steps h and h/2 and comparing the answers on the basis of the theory results Wen can estimate the error in either result

Extrapolation – Simpson's rule on one interval

Let $I(Simp,h)$ be the approximate value of the integral on one interval with Simpsons rule and step h.

Let $Iexact$ be the exact value.

Then $Iexact - I(Simp,h) \approx \dfrac{-h^5}{2880} f^{(iv)}(\zeta_1)$

and $Iexact - I(Simp,\dfrac{h}{2}) \approx \dfrac{-h^5/32}{2880} f^{(iv)}(\zeta_1)$

Extrapolation – Simpson's rule on one interval

Subtracting $Iexact - I(Simp, \frac{h}{2})$ from $Iexact - I(Simp, h)$

shows that the difference between the numerical solutions

gives an estimate of the error

$$I(Simp, \frac{h}{2}) - I(Simp, h) \approx \frac{-h^5}{2880} f^{(iv)}(\zeta_1)(1 - \frac{1}{32})$$

multiplying rhs by $\frac{32}{31}$ estimates the error in $I(Simp, h)$ and

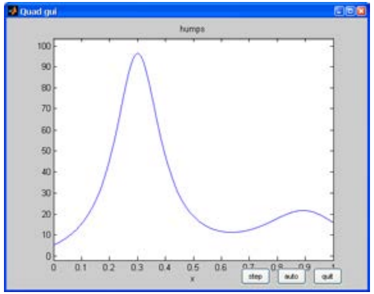
multiplying by $\frac{1}{31}$ estimates the error in $I(Simp, \frac{h}{2})$

Matlab QUADTX

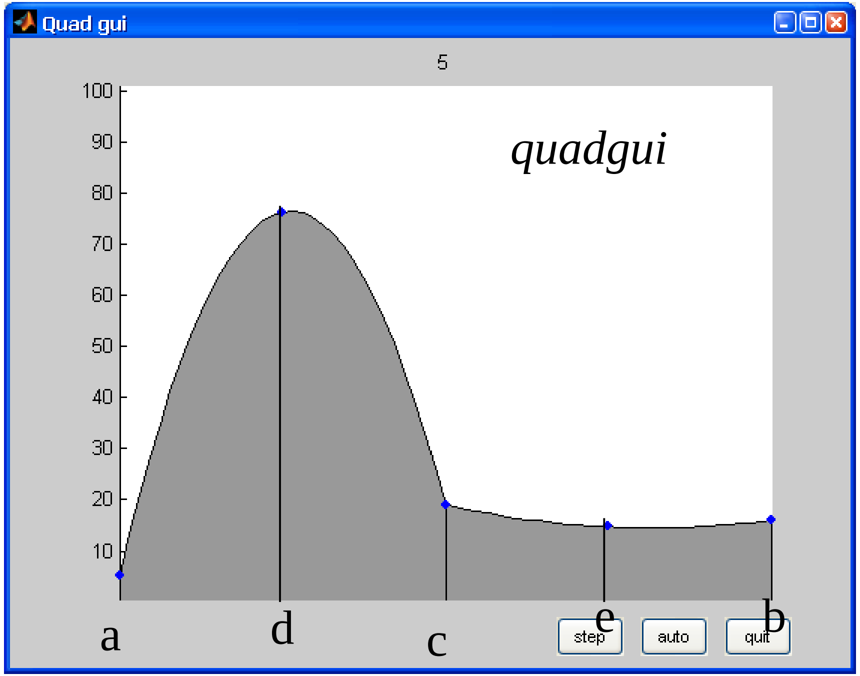
- Uses Simpson's Rule recursively until the error on each interval is less than *tol* which is user supplied
- A decision to subdivide each interval is made when the error estimated is > *tol*
- Please see the code for an example of this recursive approach

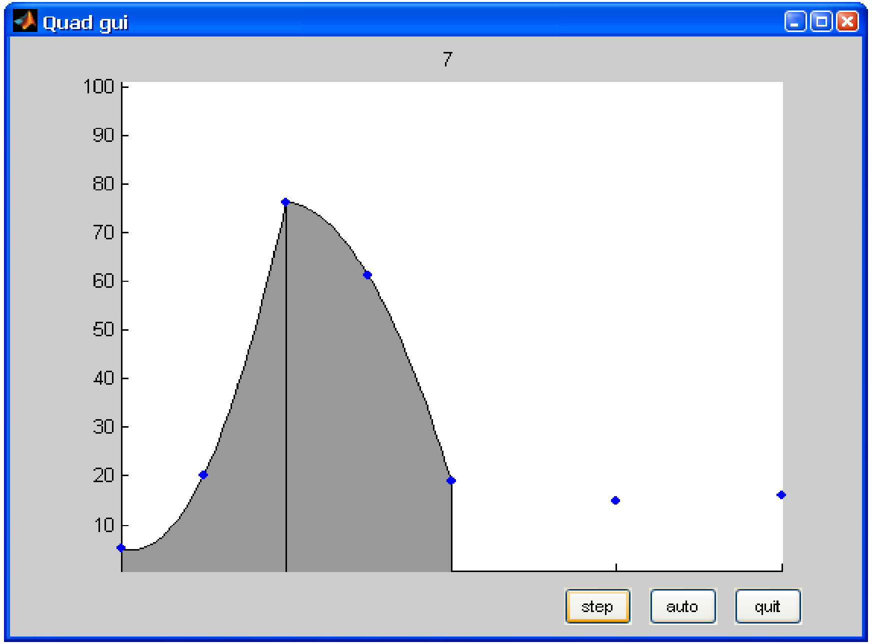
Example of Quadtx in use with the humps example

The first evaluation when a = 0 and b = 1 using Extrapolated Simpson's Rule.



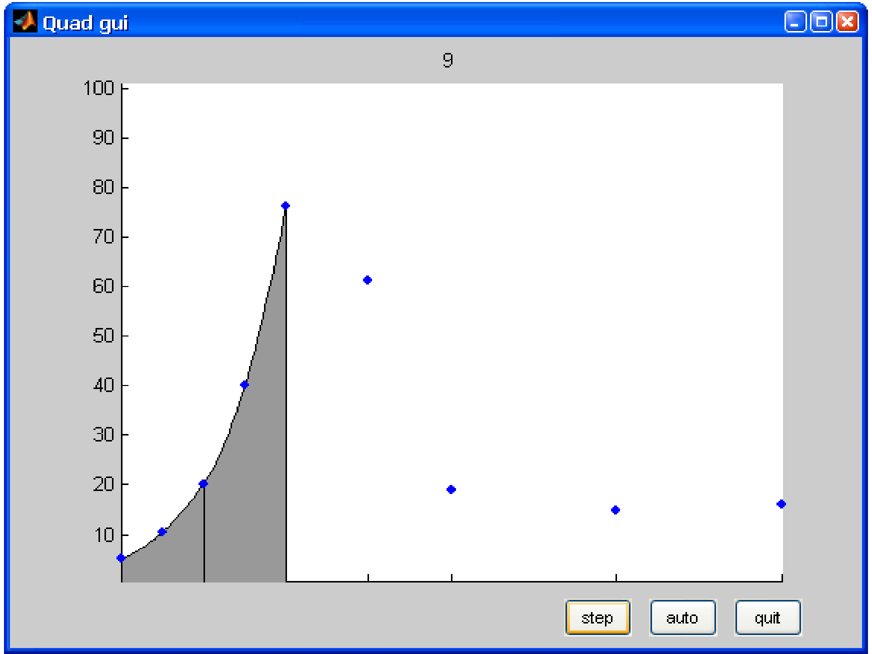
Plot of the function



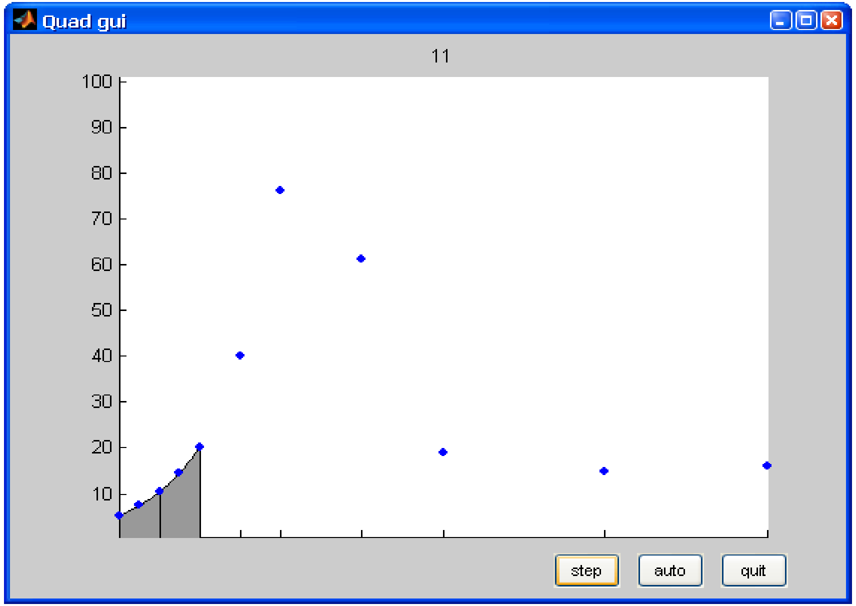


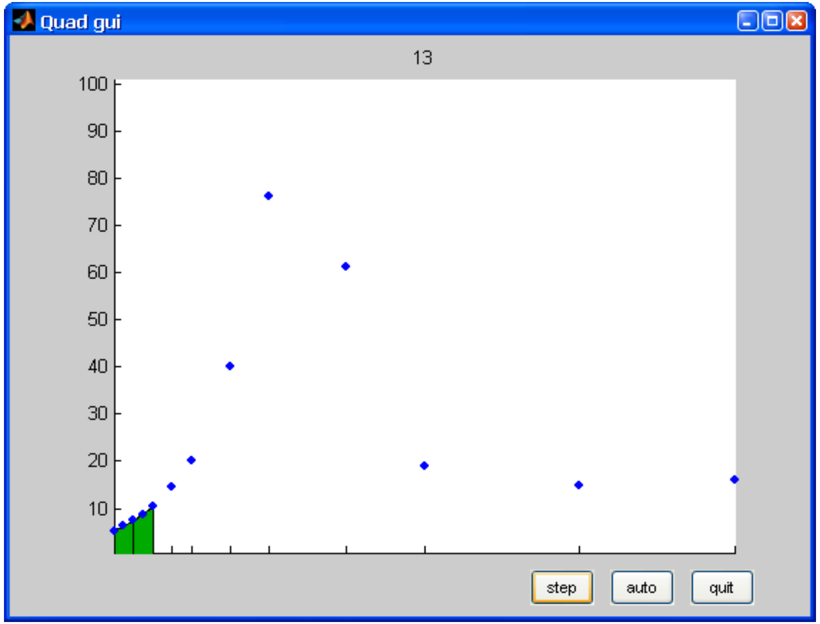
Quad gui
7
100
90
80
70
60
50
40
30
20
10
step
auto
quit

Iterations

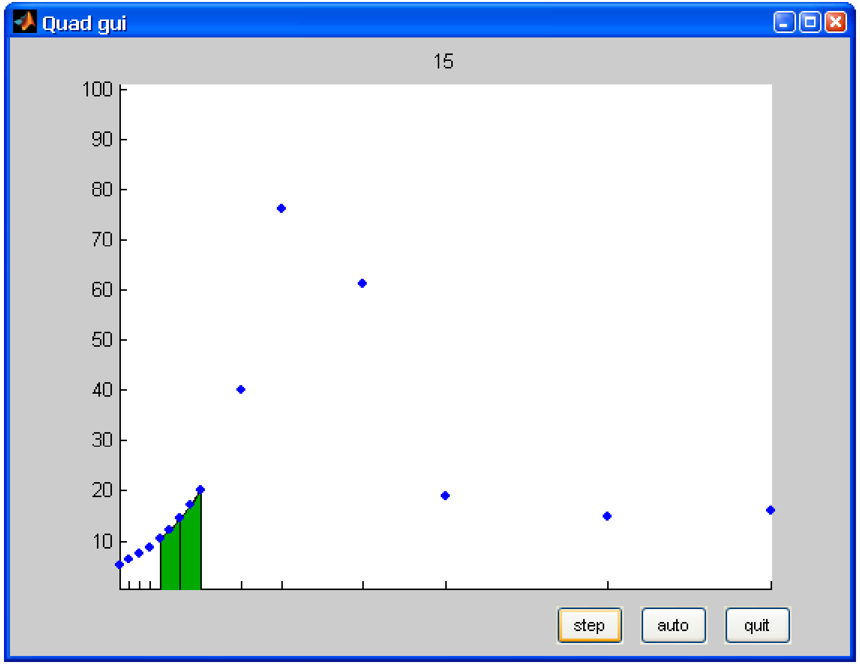


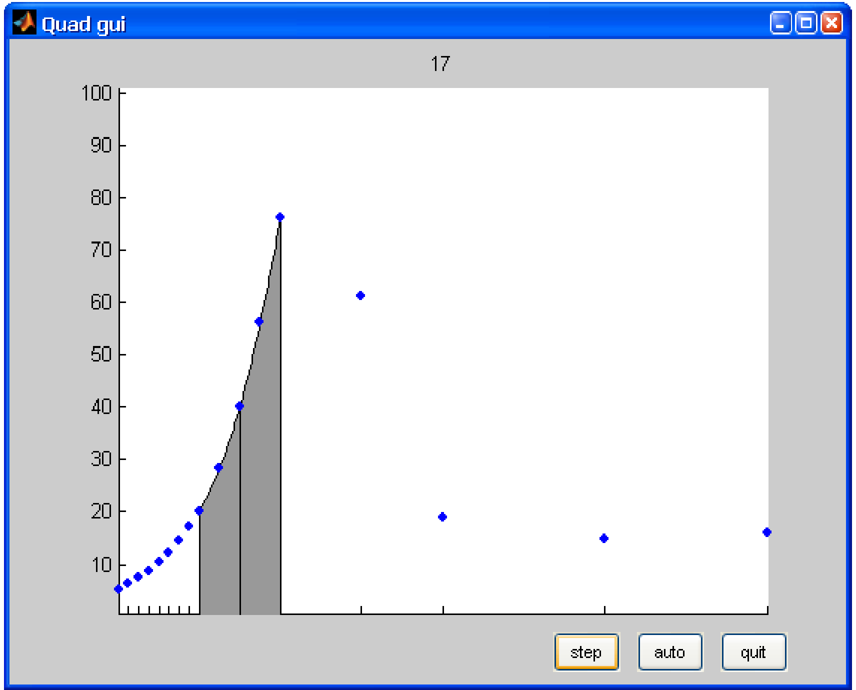
Quad gui
9
100
90
80
70
60
50
40
30
20
10
step
auto
quit


Quad gui
11
100
90
80
70
60
50
40
30
20
10
step
auto
quit

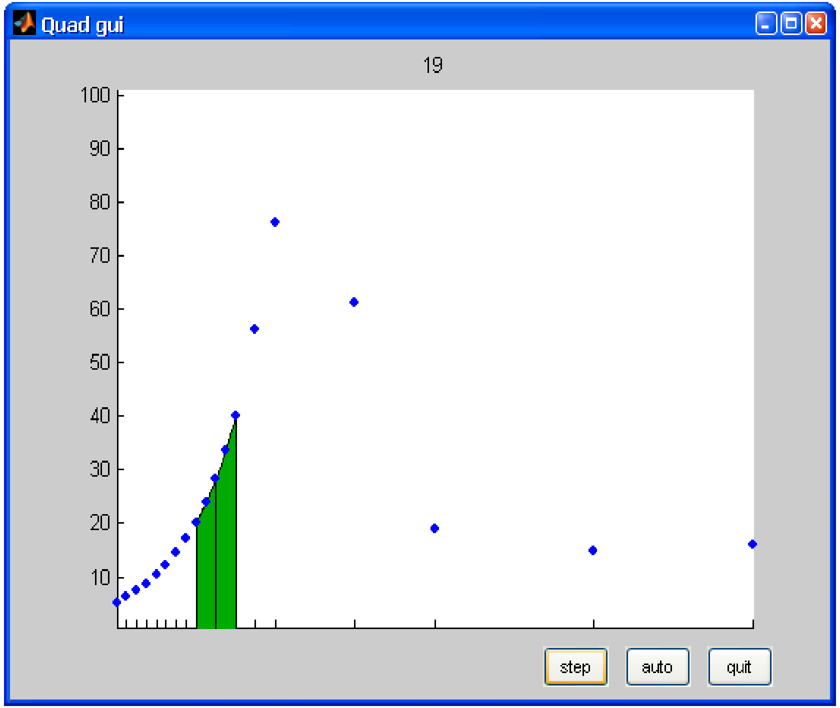

Quad gui
13
100
90
80
70
60
50
40
30
20
10
step
auto
quit

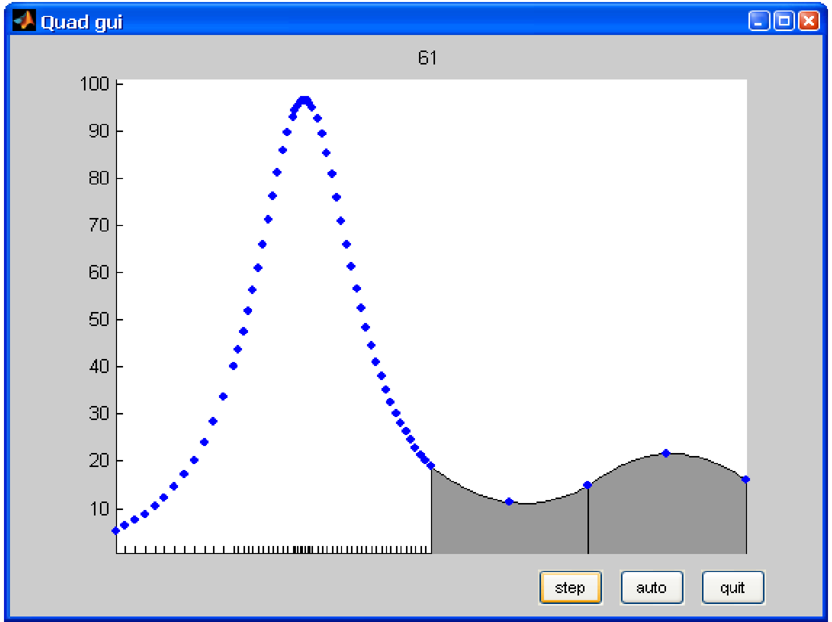
Iterations


Quad gui
15
100
90
80
70
60
50
40
30
20
10
step
auto
quit


Quad gui
17
100
90
80
70
60
50
40
30
20
10
step
auto
quit

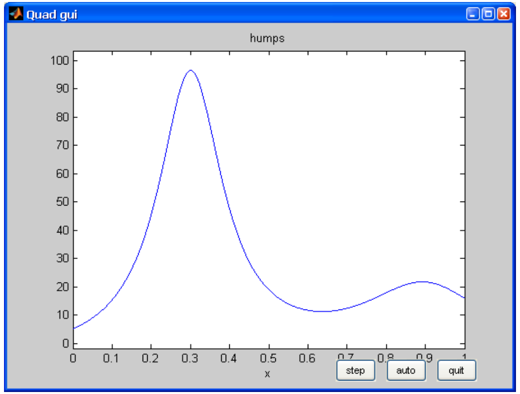
Iterations


Quad gui
19
100
90
80
70
60
50
40
30
20
10
step
auto
quit

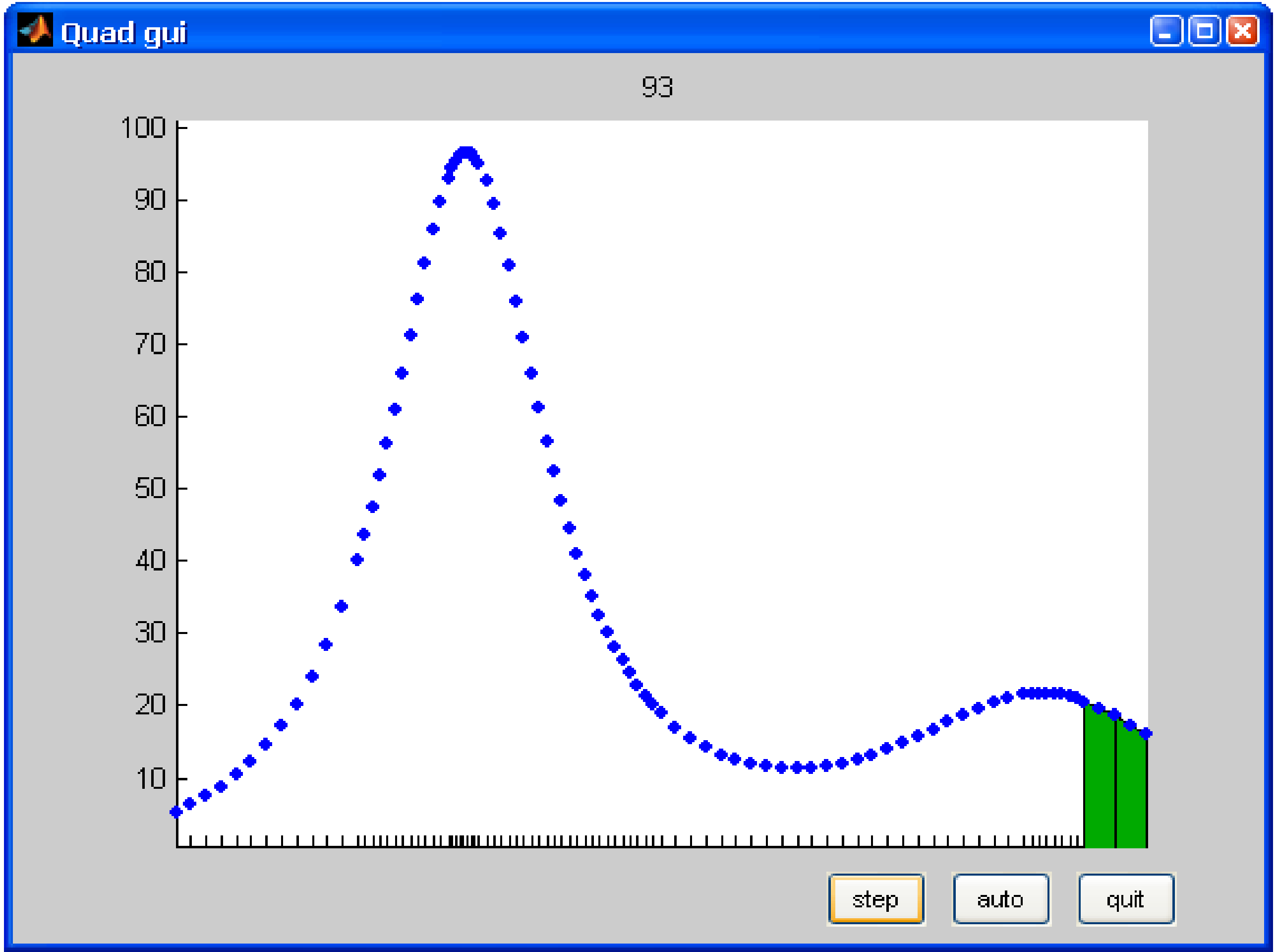

Quad gui
61
100
90
80
70
60
50
40
30
20
10
step
auto
quit

Iterations

Last Step



Plot of the function



Driving code for Humps Example

```
fprintf('  tol              Q                    fcount      err        ratio \n')
for  k = 1:12
    tol = 10^(-k);
    Qexact=29.85832539549867;
    [Q,fcount] = quadtx(@humps,0,1,tol);
    err=Q-Qexact;
    ratio = err/tol;
    fprintf('%8.0e %21.14f %7d %13.3e %9.3f \n',tol,Q,fcount,err,ratio)
end
```

Results for Humps Example

tol	Q	fcount	err	ratio
1e-01	29.83328444174864	25	-2.504e-02	-0.250
1e-02	29.85791444629948	41	-4.109e-04	-0.041
1e-03	29.85834299237637	69	1.760e-05	0.018
1e-04	29.85832444437543	93	-9.511e-07	-0.010
1e-05	29.85832551548643	149	1.200e-07	0.012
1e-06	29.85832540194041	265	6.442e-09	0.006
1e-07	29.85832539499819	369	-5.005e-10	-0.005
1e-08	29.85832539552631	605	2.764e-11	0.003
1e-09	29.85832539549604	1061	-2.636e-12	-0.003
1e-10	29.85832539549890	1469	2.309e-13	0.002
1e-11	29.85832539549867	2429	-3.553e-15	-0.000
1e-12	29.85832539549867	4245	3.553e-15	0.004

Use of adaptive methods

- Many similar examples in interpolation, quadrature and solution of differential equations
- More challenging to implement and run especially in parallel as we do not know where the work will be in advance.
- Current state of the art is that these kind of methods run on the very largest computers e.g. the Uintah code developed here.