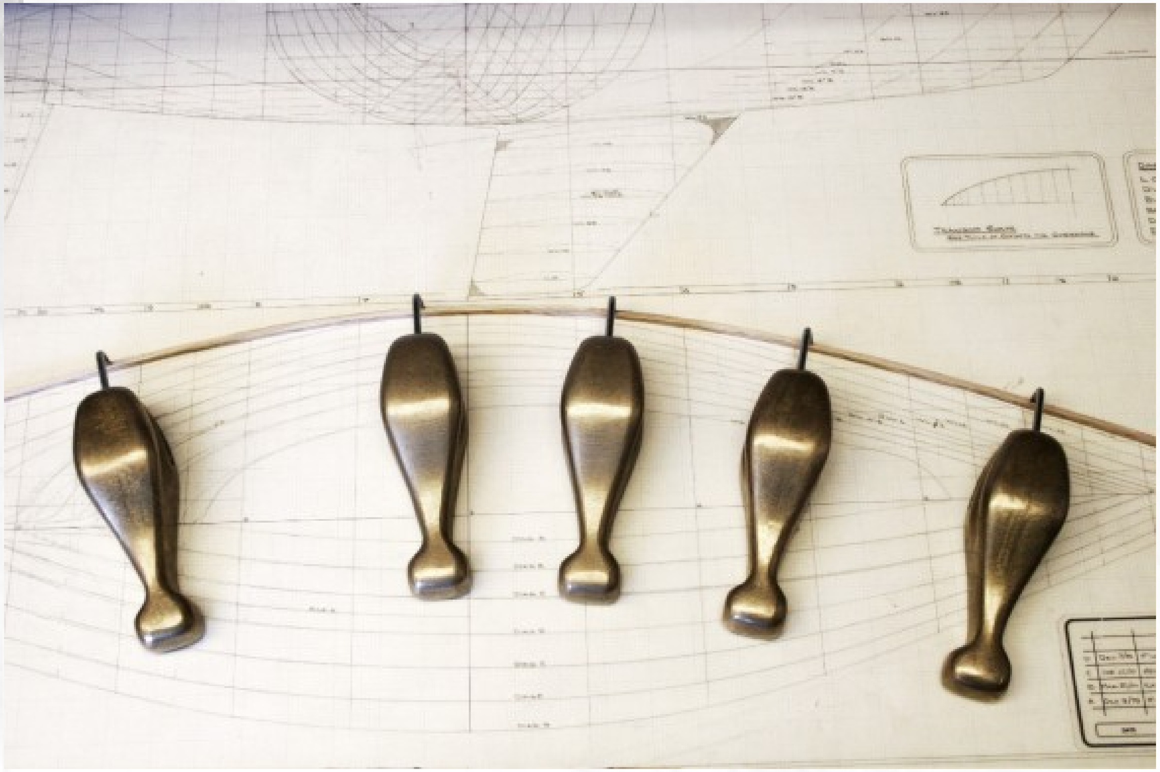
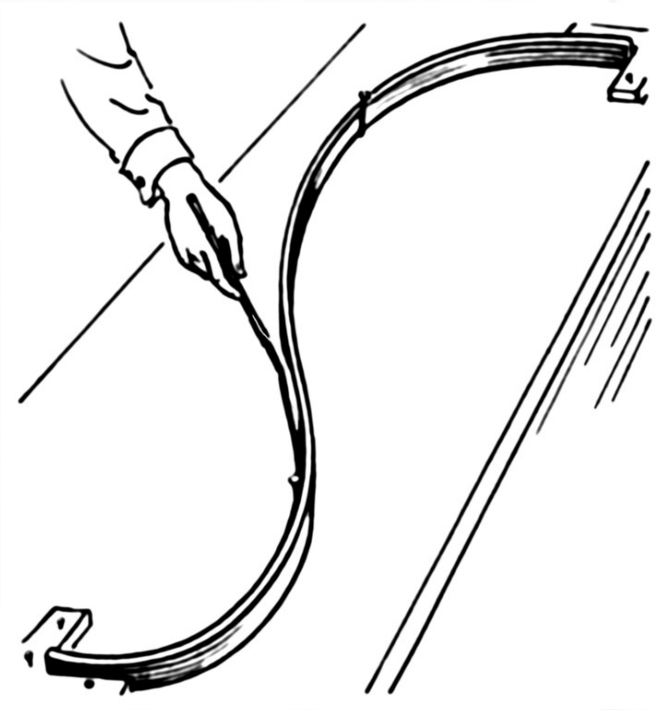
New topic: 1D Splines

- Recall problem with 1D interpolation: Runge phenomenon.
- Solution: Don't interpolate an entire region of with one curve. Rather, just interpolate short regions with short curves, $s_n(x)$, then stitch curves together.
- Must match boundary conditions at joint between regions: Match curves, $s_n(x) = s_{n+1}(x)$, and 1st & 2nd derivatives.

Spline

- Drawing tool used in ship building for centuries

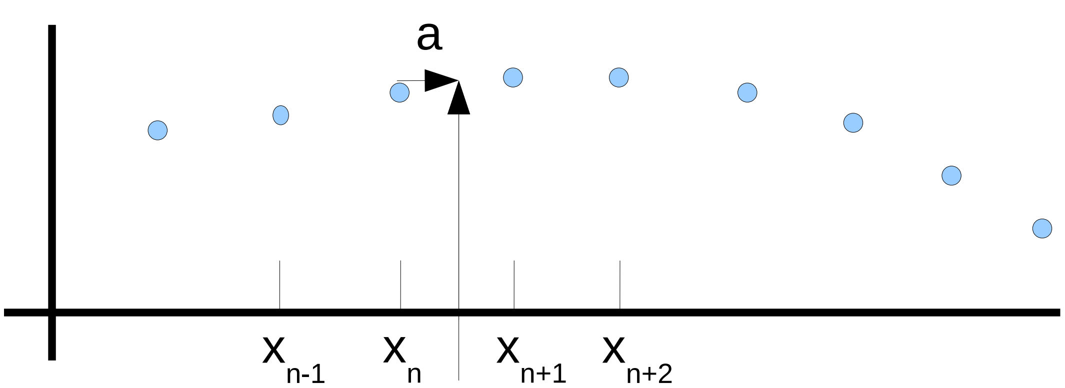




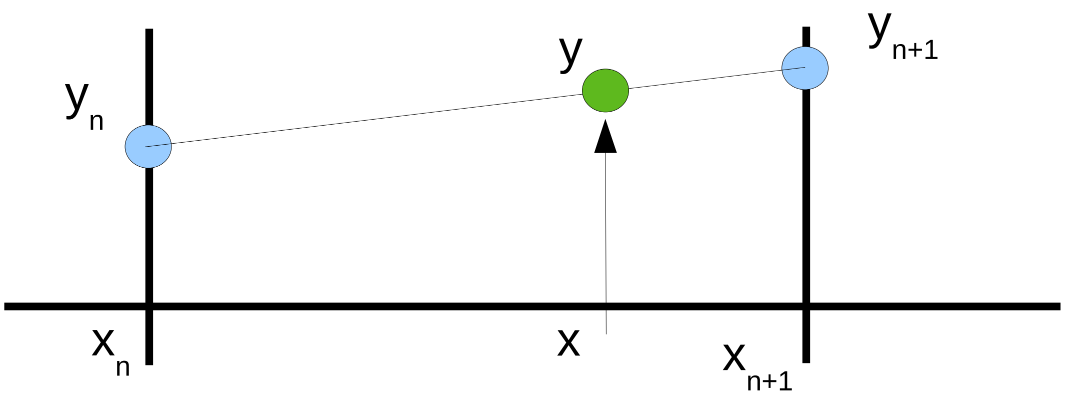
- Modern mathematical spline mimics the behavior of the drawing aid – stiffness is used to create smooth curve.

Recall linear interpolation

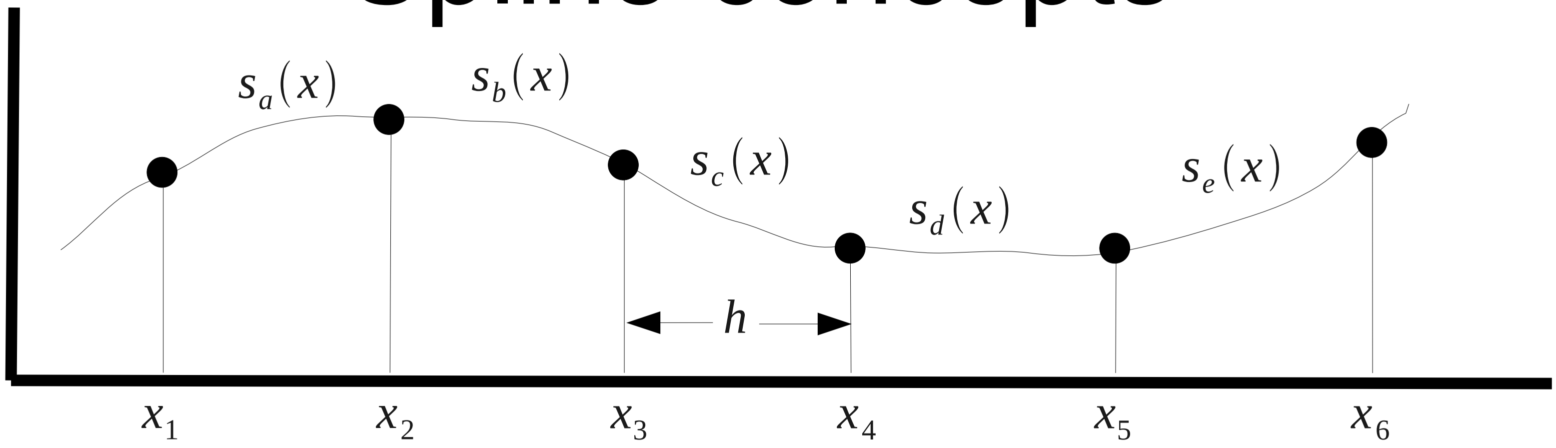
- Treat every interval independently



- Interpolate inside interval using some function.



Spline concepts



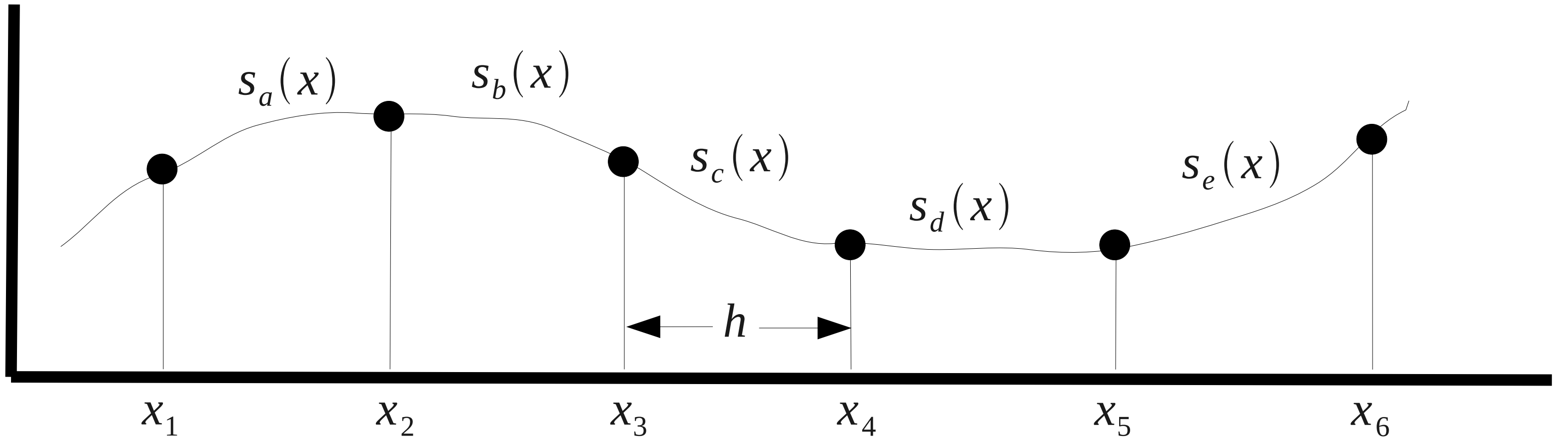
- Use different poly in each interval
- In general, people use 3rd order spline.
- Connection points are called “knots”.
- Goal: fit 3rd degree polys to satisfy at knots:

$s_a(x_n) = s_b(x_n)$ Fit values at boundaries

$s_a'(x_n) = s_b'(x_n)$ Fit slopes at boundaries

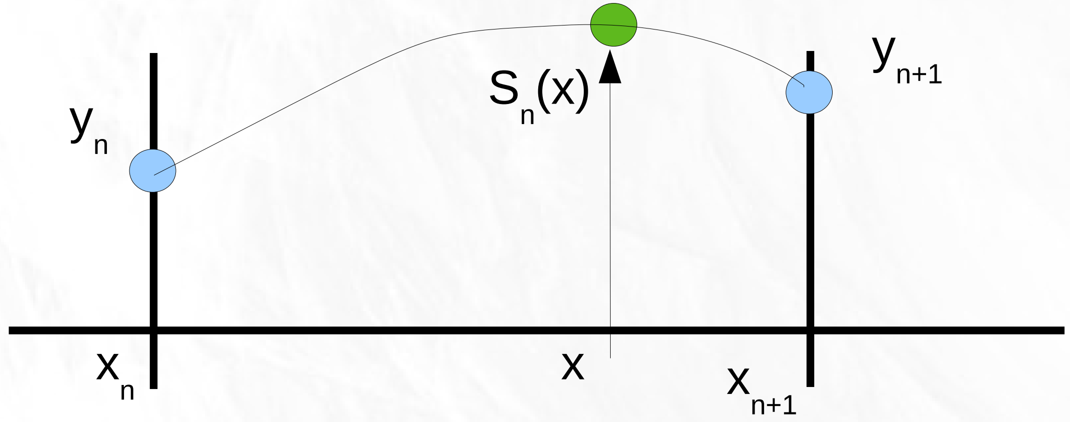
$s_a''(x_n) = s_b''(x_n)$ Fit 2nd derivs at boundaries

Two free parameters at spline end



- Each 3rd degree poly has 4 free parameters. 5 polys => 20 total free parameters.
- Each knot introduces 3 constraints. 6 knots => 18 equations.
- 18 equations and 20 unknowns => we have 2 free parameters.
  - Free parameters are fixed using conditions set at ends of domain.

Derivation of spline equations



- Define $S_n(x)$ as 3rd degree polynomial

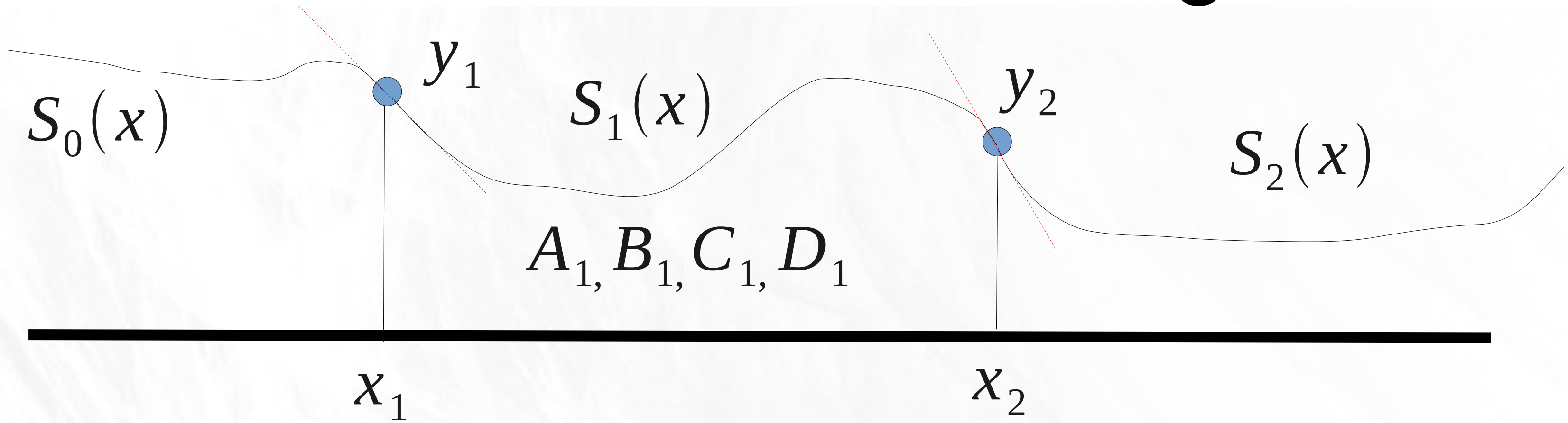
$$S_n(x) = A_n + B_n(x - x_n) + C_n(x - x_n)^2 + D_n(x - x_n)^3$$

- Derivatives are

$$S_n'(x) = B_n + 2C_n(x - x_n) + 3D_n(x - x_n)^2$$

$$S_n''(x) = 2C_n + 6D_n(x - x_n)$$

Consider one interior segment, two knots and two exterior segment



- Curve $S_1(x)$ is fixed at x_1, x_2.
- Consider slopes $S'_0(x_1)$ and $S'_2(x_2)$ known and fixed -- "clamped".
- Therefore, we have 4 eqs and 4 unks (A_1, B_1, C_1, D_1).

$$S_1(x_1)=y_1$$
$$\Rightarrow \quad A_1=y_1$$

$$S_1(x_2)=y_2$$
$$\Rightarrow \quad A_1+B_1h+C_1h^2+D_1h^3=y_2$$

$$S_1'(x_1)=S_0'(x_1)$$
$$\Rightarrow \quad B_1=S_0'(x_1)$$

$$S_1'(x_2)=S_2'(x_2)$$
$$\Rightarrow \quad B_1+2C_1h+3D_1h^2=S_2'(x_2)$$

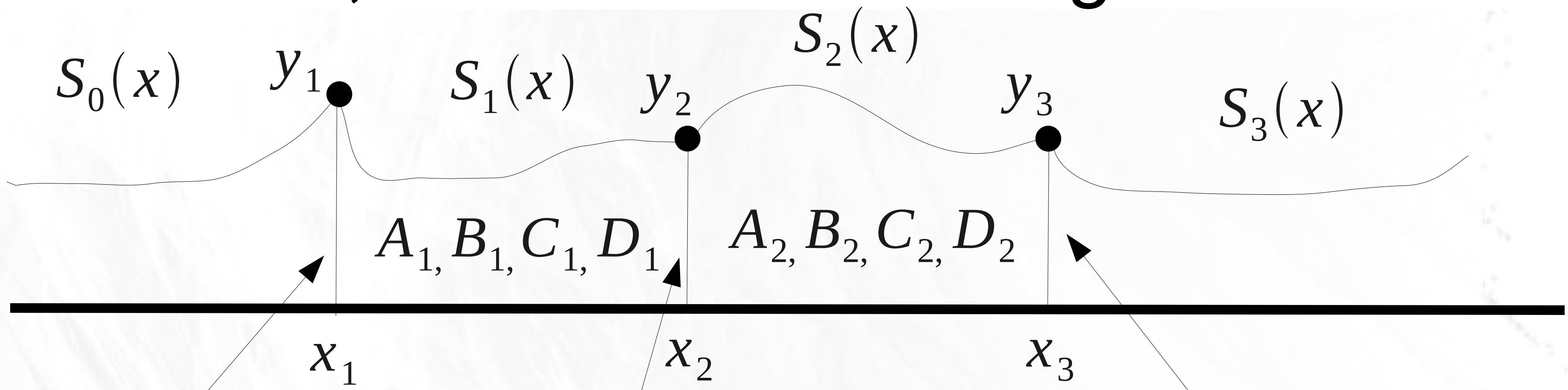
Written in matrix format

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & h & h^2 & h^3 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2h & 3h^2 \end{pmatrix} \begin{pmatrix} A_1 \\ B_1 \\ C_1 \\ D_1 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ S_0'(x_1) \\ S_2'(x_2) \end{pmatrix}$$

Exterior slopes are arbitrary – set them to do something sensible for your problem.

- Solve this equation to get A, B, C, D for the spline polynomial in this segment.

Now consider two interior segments, three knots, and two exterior segments



Two eqs --
Match y1 value for S1
Match 1st derivatives of S1, S0

Four eqs --
Match y2 value for S1
Match y2 value for S2
Match 1st derivative of S1, S2
Match 2nd deriv of S1, S2

Two eqs --
Match y3 value for S2
Match 1st derivatives of S2, S3

- Four equations and four unknowns: A_1, B_1, C_1, D_1, A_2, B_2, C_2, D_2.

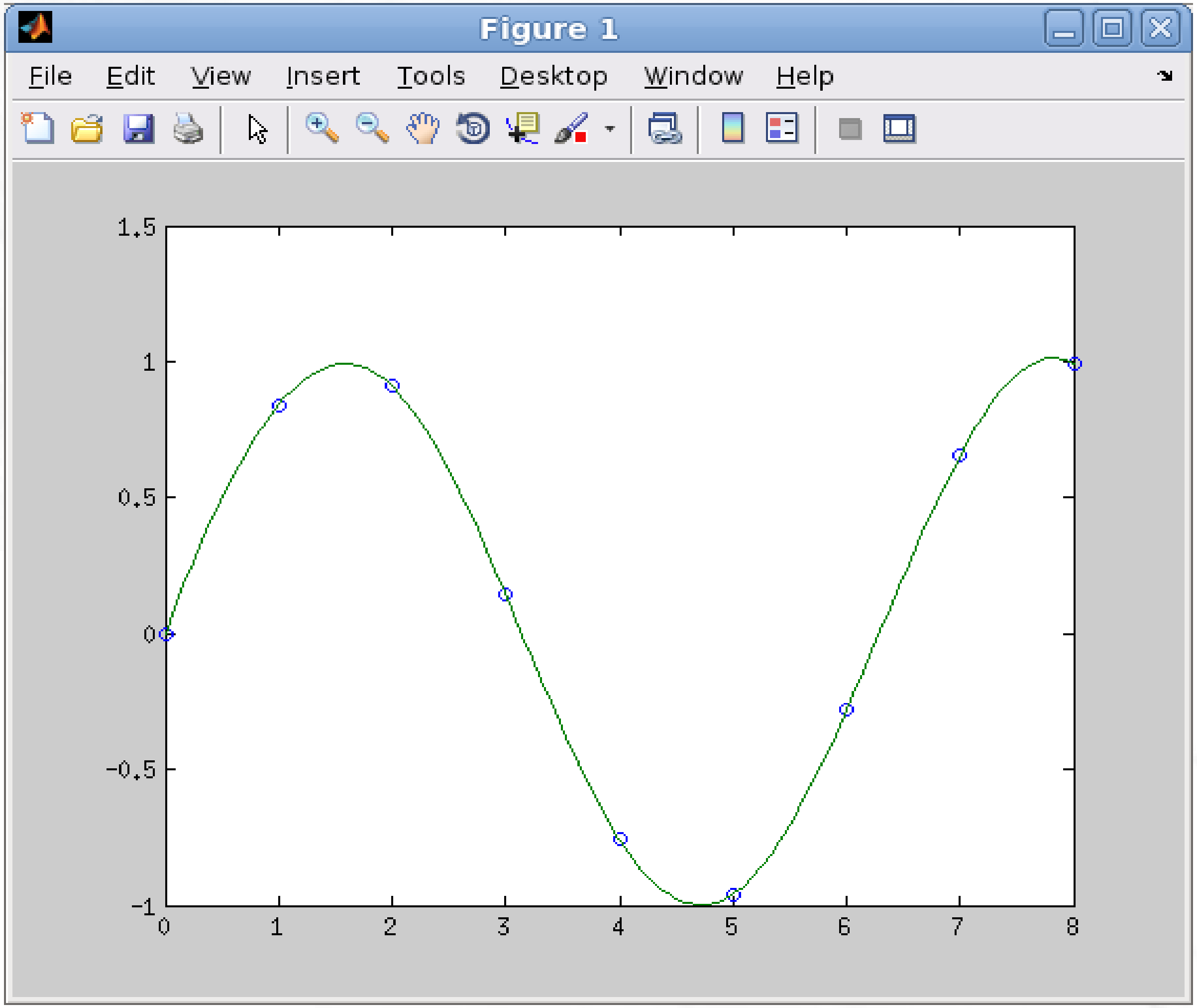
Written in matrix format

$$
\left(\begin{array}{cccc|cccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & h & h^2 & h^3 & -1 & 0 & 0 & 0 \\
\hline
0 & 1 & 2h & 3h^2 & 0 & -1 & 0 & 0 \\
0 & 0 & 2 & 6h & 0 & 0 & -2 & 0 \\
0 & 0 & 0 & 0 & 1 & h & h^2 & h^3 \\
0 & 0 & 0 & 0 & 0 & 1 & 2h & 3h^2
\end{array}\right)
\left(\begin{array}{c}
A_1 \\ B_1 \\ C_1 \\ D_1 \\ A_2 \\ B_2 \\ C_2 \\ D_2
\end{array}\right)
=
\left(\begin{array}{c}
y_1 \\ S_0'(x_1) \\ y_2 \\ 0 \\ 0 \\ 0 \\ y_3 \\ S_3'(x_3)
\end{array}\right)
$$

- Problem reduces to solving an ugly Ax = b problem.
- The thing to solve for are the A, B, C, D coefficients of each 3rd degree polynomial describing each segment.
- Matlab provides built-in functions for this.

Matlab

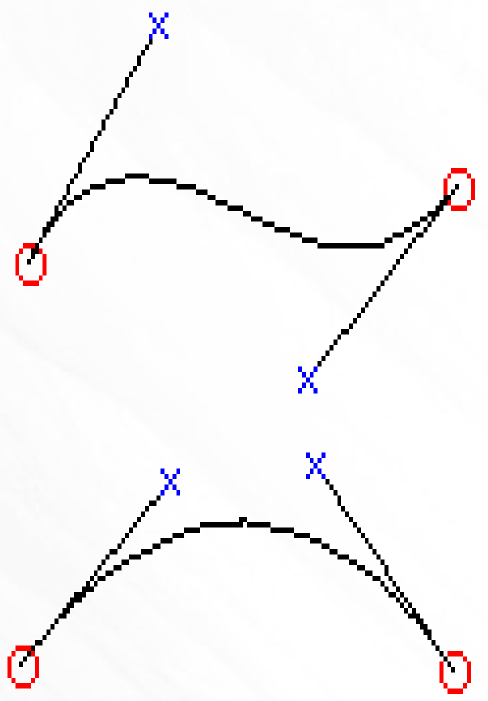
- Matlab built-in: spline.
- Returns polynomial object
- Call ppval to evaluate polynomial object.




```
>> x = 0:8;
>> y = sin(x);
>> pp = spline(x, y)

pp =

      form: 'pp'
    breaks: [0 1 2 3 4 5 6 7 8]
     coefs: [8x4 double]
    pieces: 8
     order: 4
       dim: 1

>> xx = linspace(0, 8, 100);
>> plot(x, y, 'o', xx, ppval(pp, xx), '-');
```

Spline interpolation

- Many different types of spline using different polynomials
  - 3^{rd} degree polynomial is the classic one. Matlab: spline. Returns polynomial coeffs.
  - Cubic Hermite polynomial. Matlab: pchip.
  - Bezier.   Good mini-project topic
  - Others....

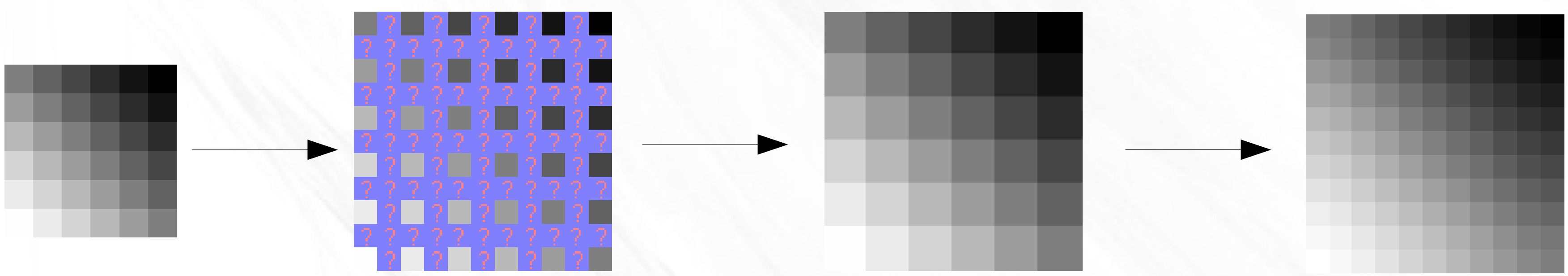


Bezier splines

Interpolation in 2 dimensions

New topic: 2D interpolation (image processing)

- Important in re-sizing images
  - Bilinear interpolation (2D images)
  - Bicubic interpolation



Original image | Expanded | Expanded pixels | Interpolated

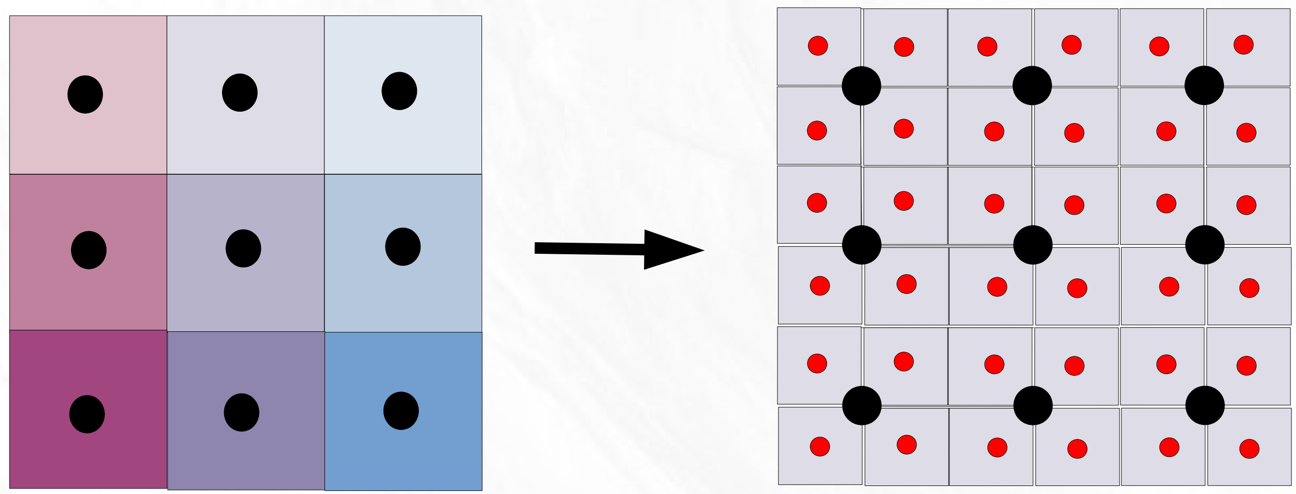
Interpolation in image processing

Original image
(has “the jaggies”)

Upsampled image
after interpolation

Upsampling an image

Original image 2x upsampled image

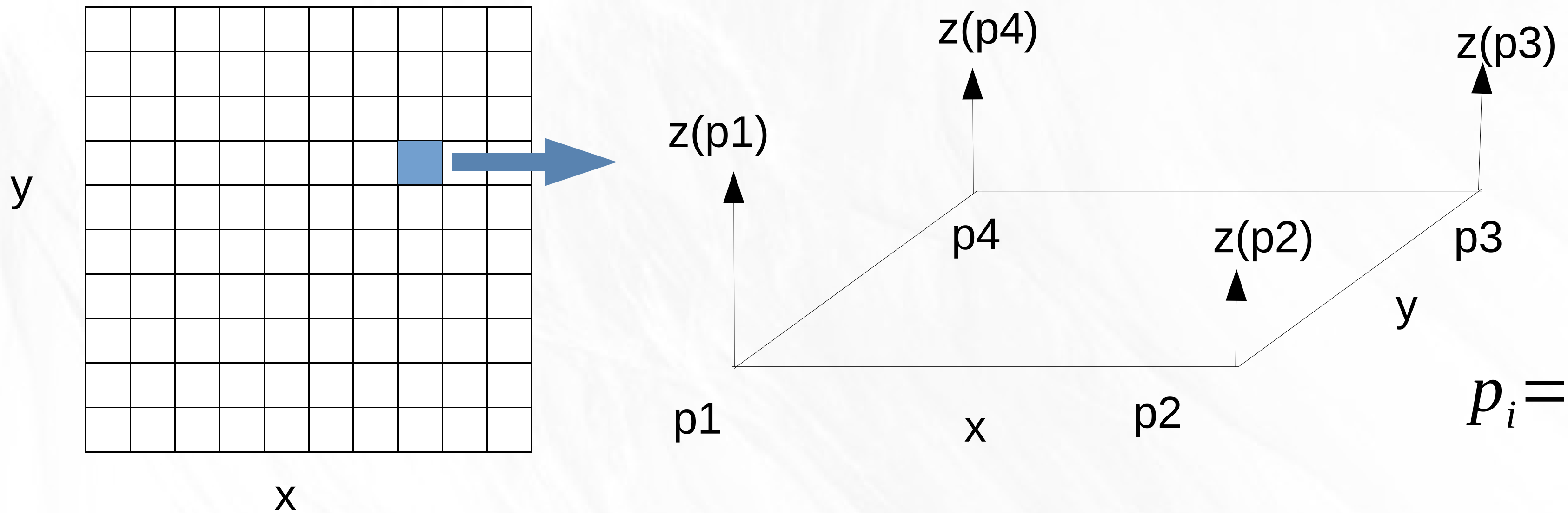


Pixel values held in center of large squares (black dots)

Pixel values held in center of small squares (red dots)

How to get values at red dots?

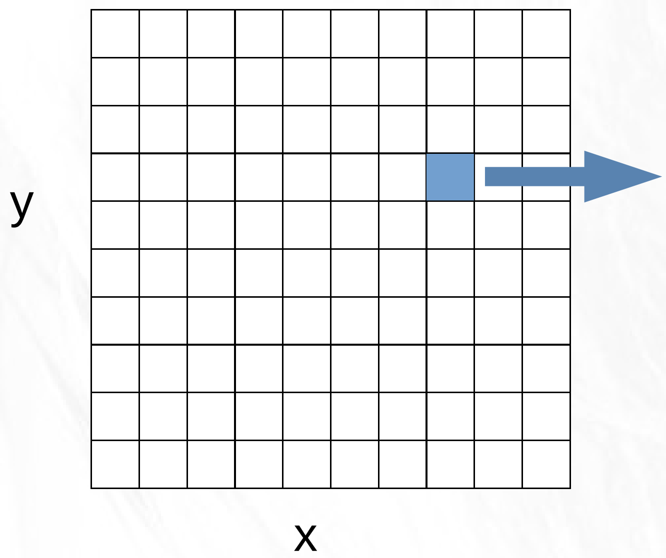
Simple: Bilinear interpolation

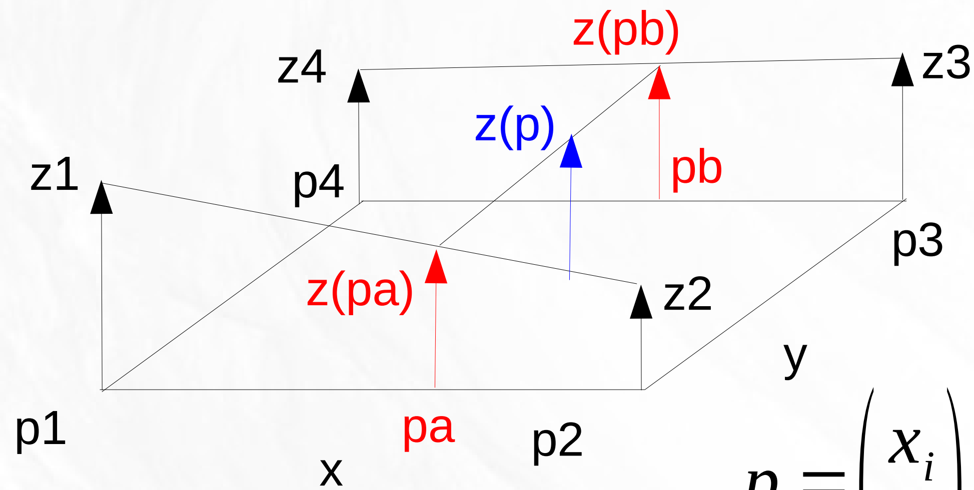


$$p_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix}$$

- The function to interpolate is split up into squares.
- In each square, we know the values of z at four corners p1, p2, p3, p4.
- We want to interpolate to get z inside the square.
- Three points define a plane, but we have 4 points!

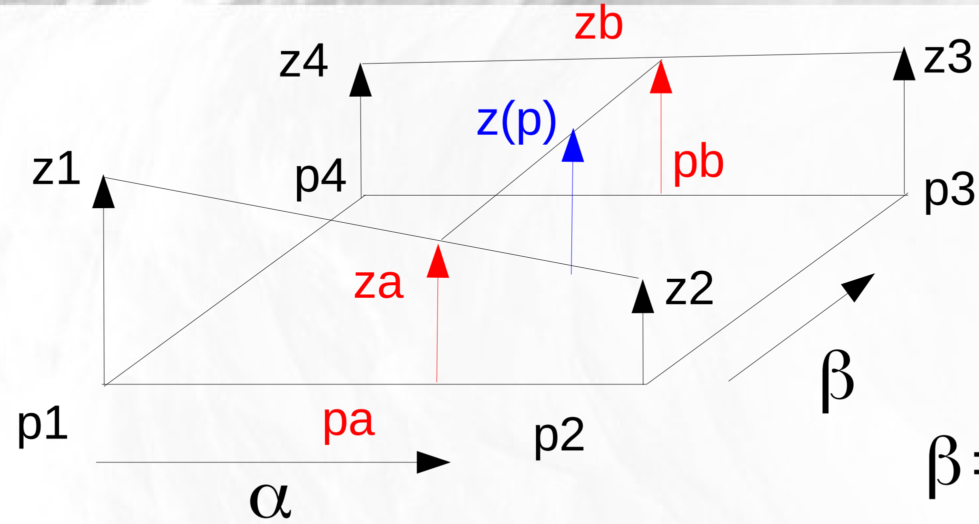
Algorithm: construct three lines...





$$p_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix}$$

1. First figure out which square you are in.
2. Do linear interpolation between z1 and z2 to get za
3. Do linear interpolation between z3 and z4 to get zb
4. Do linear interpolation between za and zb to get z(p)



$$\beta = \frac{y - y_1}{y_4 - y_1}$$

$$\alpha = \frac{x - x_1}{x_2 - x_1}$$

$$z_a = (1-\alpha) z_1 + \alpha z_2$$

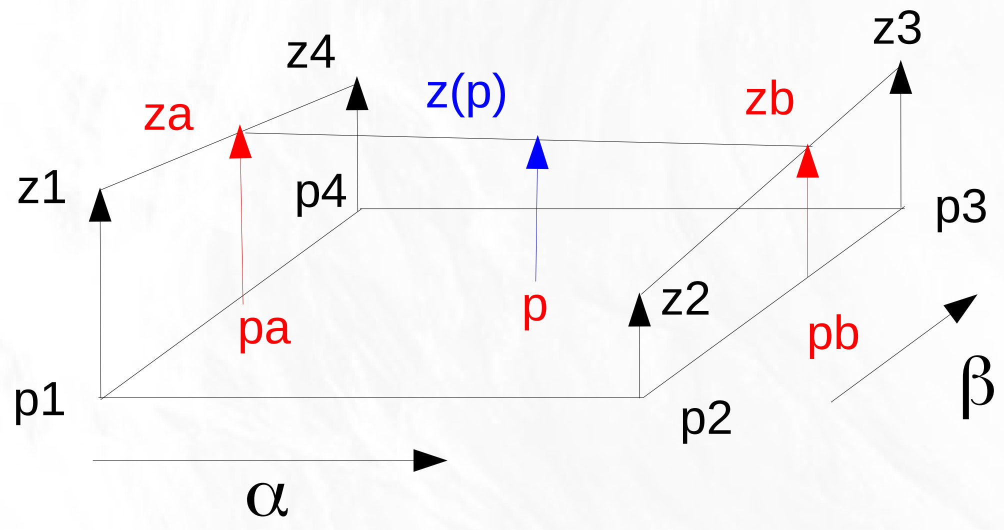
$$z_b = (1-\alpha) z_4 + \alpha z_3$$

Note this is second degree. Second degree terms are $\alpha\beta$

$$z(p) = (1-\beta) z_a + \beta z_b$$

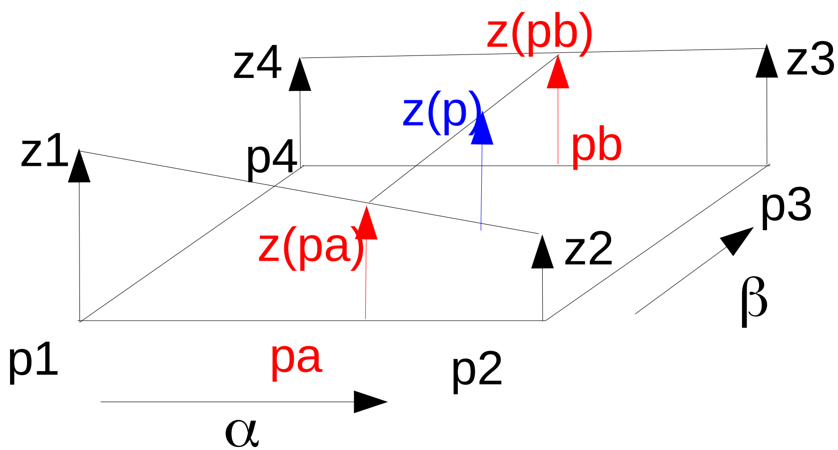
$$= (1-\beta)(1-\alpha) z_1 + (1-\beta)\alpha z_2 + \beta(1-\alpha) z_4 + \beta\alpha z_3$$

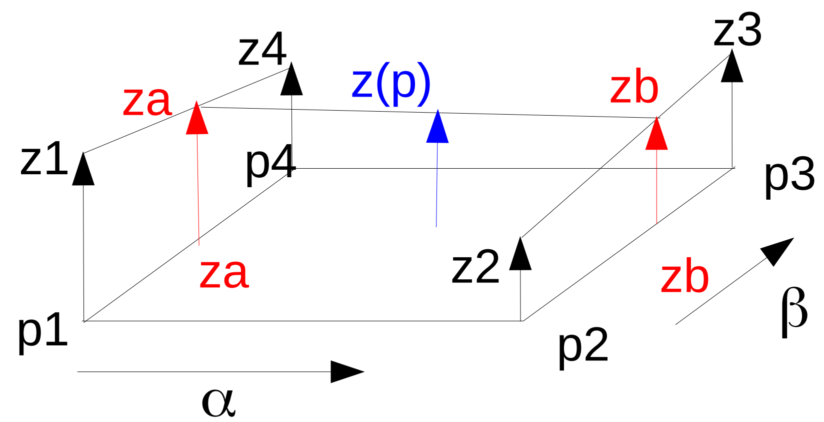
It also works the other way...



1. Do linear interpolation between p1 and p4 to get za
2. Do linear interpolation between p2 and p3 to get zb
3. Do linear interpolation between za and zb to get z(p)

It works both ways -- proof





$$\alpha = \frac{x - x_1}{x_2 - x_1} \qquad \beta = \frac{y - y_1}{y_4 - y_1}$$

$$z_a = (1-\alpha) z_1 + \alpha z_2$$

$$z_b = (1-\alpha) z_4 + \alpha z_3$$

$$z(p) = (1-\beta) z_a + \beta z_b$$

$$= (1-\beta)(1-\alpha) z_1 + (1-\beta)\alpha z_2 + \beta(1-\alpha) z_4 + \beta \alpha z_3$$

$$\alpha = \frac{x - x_1}{x_2 - x_1} \qquad \beta = \frac{y - y_1}{y_4 - y_1}$$

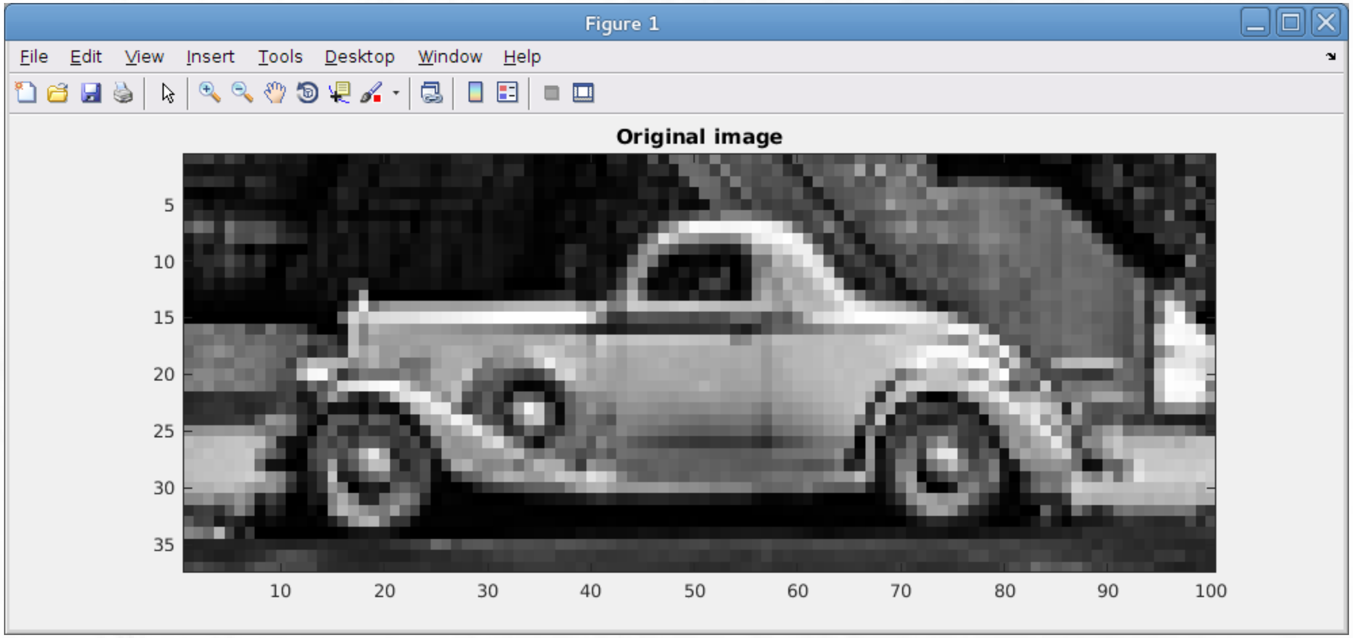
$$z_a = (1-\beta) z_1 + \beta z_4$$

$$z_b = (1-\beta) z_2 + \beta z_3$$

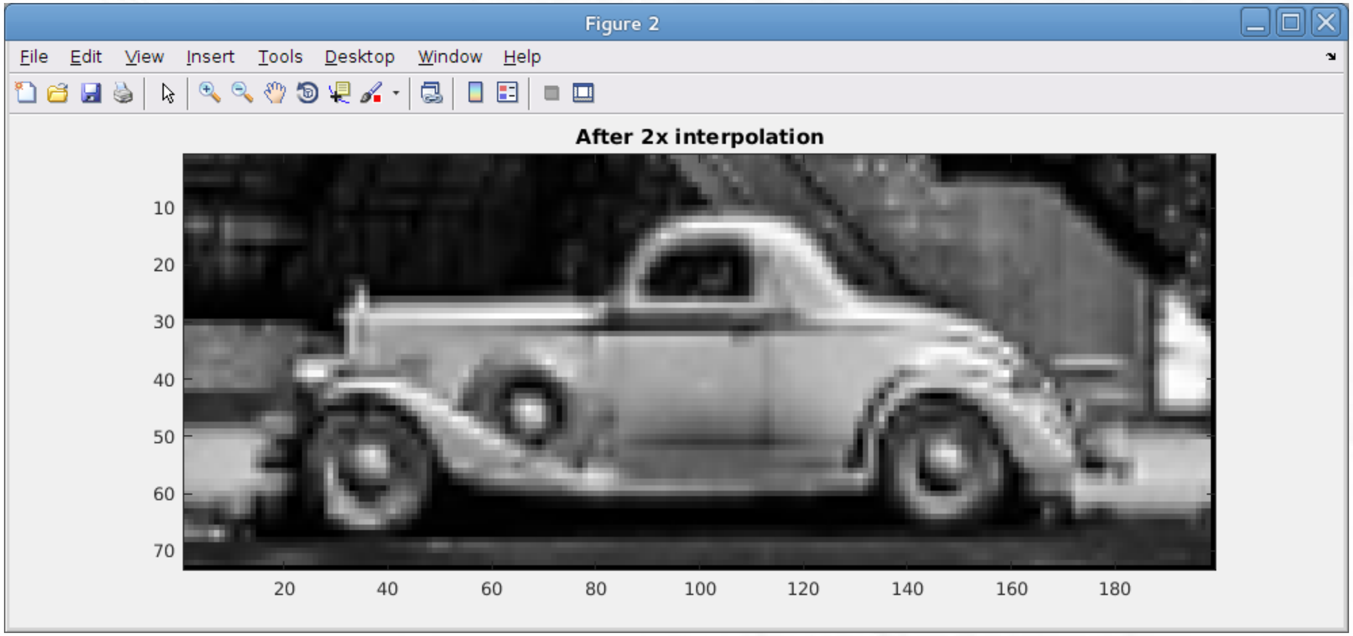
$$z(p) = (1-\alpha) z_a + \alpha z_b$$

$$= (1-\alpha)(1-\beta) z_1 + (1-\alpha)\beta z_4 + \alpha(1-\beta) z_2 + \alpha \beta z_3$$

Example image interpolation

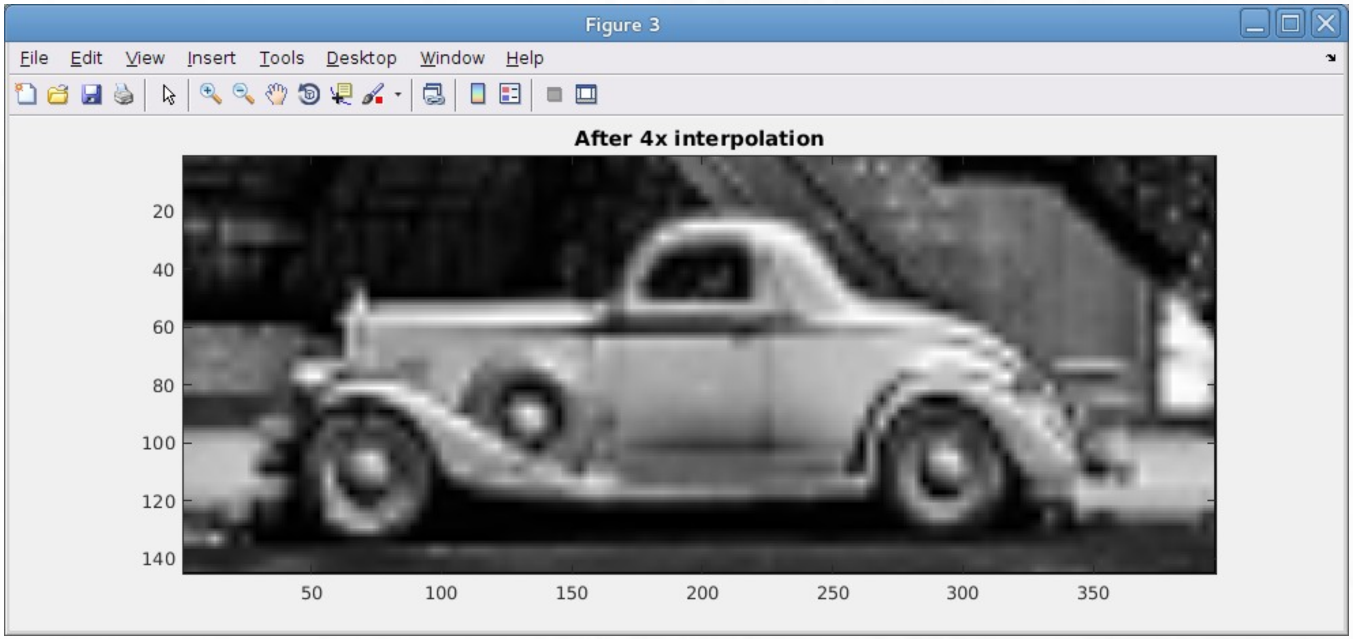


Original

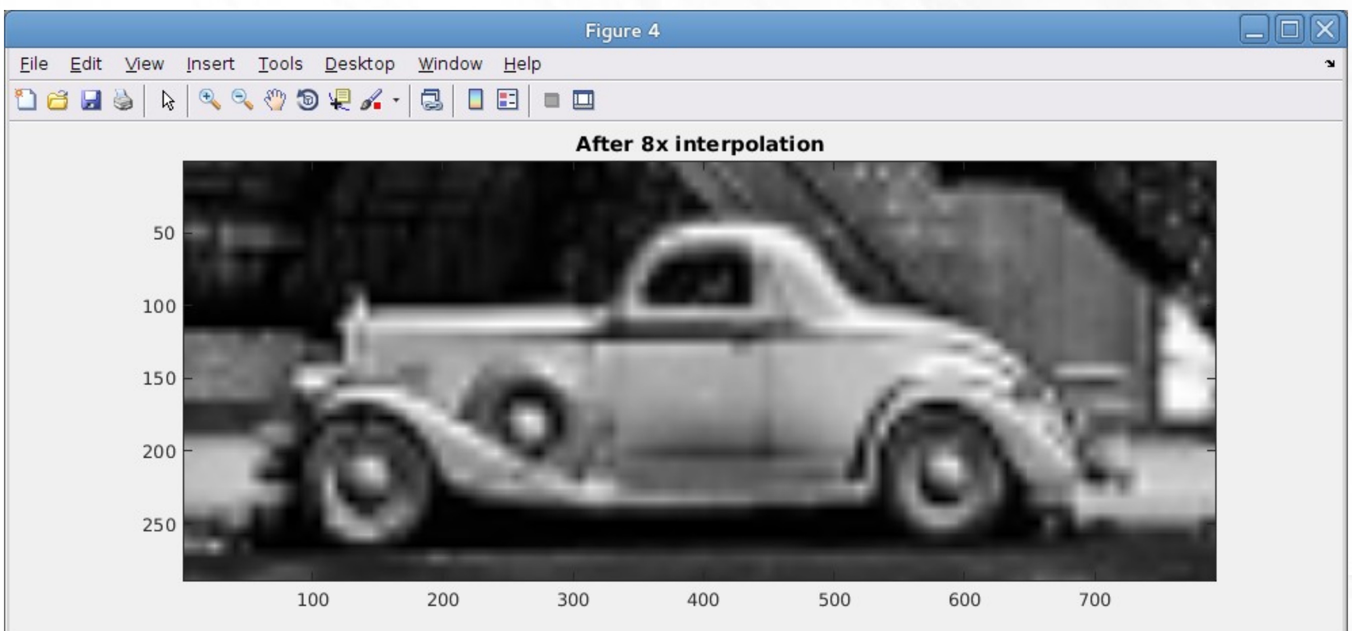


2x Bilinear Interpolation

/home/sdb/Northeastern/Class9/Bilinear



4x Bilinear Interpolation



8x Bilinear Interpolation

Order of interpolation?

- You can't fit a plane to 4 points (in general).
- Recall expression for z(p):

$$z(p)=(1-\beta)(1-\alpha)z_1+(1-\beta)\alpha z_2+\beta(1-\alpha)z_4+\beta\alpha z_3$$

- This expression has 2nd degree terms in $\alpha\beta$
- This suggests we are actually interpolating with a polynomial of form

Recall these are proxys for x, y.

$$z(x,y)=A+Bx+Cy+Dxy$$

How to get A, B, C, D?

- Consider interpolating polynomial

$$z(x, y) = A + Bx + Cy + Dxy$$

A, B, C, D are unknowns.

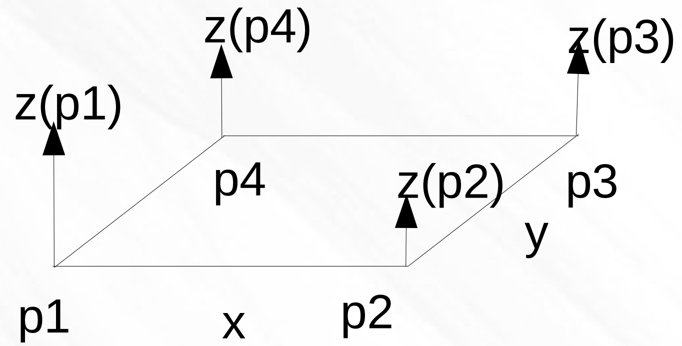
- This polynomial must interpolate z at corners exactly:

$$z_1 = A + Bx_1 + Cy_1 + Dx_1y_1$$

$$z_2 = A + Bx_2 + Cy_2 + Dx_2y_2$$

$$z_3 = A + Bx_3 + Cy_3 + Dx_3y_3$$

$$z_4 = A + Bx_4 + Cy_4 + Dx_4y_4$$



$$p_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix}$$

A, B, C, D are unknowns.

Smells like a matrix solve problem...

Bilinear interpolation using matrix

- Get interpolation coefficients A, B, C, D by solving linear system

Known [x, y] coordinates of corners

Known z values at corners

$$\begin{pmatrix} 1 & x_1 & y_1 & x_1 y_1 \\ 1 & x_2 & y_2 & x_2 y_2 \\ 1 & x_3 & y_3 & x_3 y_3 \\ 1 & x_4 & y_4 & x_4 y_4 \end{pmatrix} \begin{pmatrix} A \\ B \\ C \\ D \end{pmatrix} = \begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{pmatrix}$$

- Perform interpolation at arbitrary (x, y) using polynomial

$$z(x, y) = A + B x + C y + D x y$$

Do this many times, once for each input [x,y]

Do this once.

- Note this admits OO approach:
  - Constructor creates interpolation matrix, does solve, gets A, B, C, D.
  - Interpolate method evaluates polynomial for each [x,y].

How does the matrix method relate to the previous line method?

- Line method

Inputs: [x, y], Coordinates of interpolation box.

$$\alpha = \frac{x - x_1}{x_2 - x_1} \qquad \beta = \frac{y - y_1}{y_4 - y_1}$$

$$z_a = (1-\alpha) z_1 + \alpha z_2$$

$$z_b = (1-\alpha) z_4 + \alpha z_3$$

$$z(p) = (1-\beta) z_a + \beta z_b$$

- Matrix method

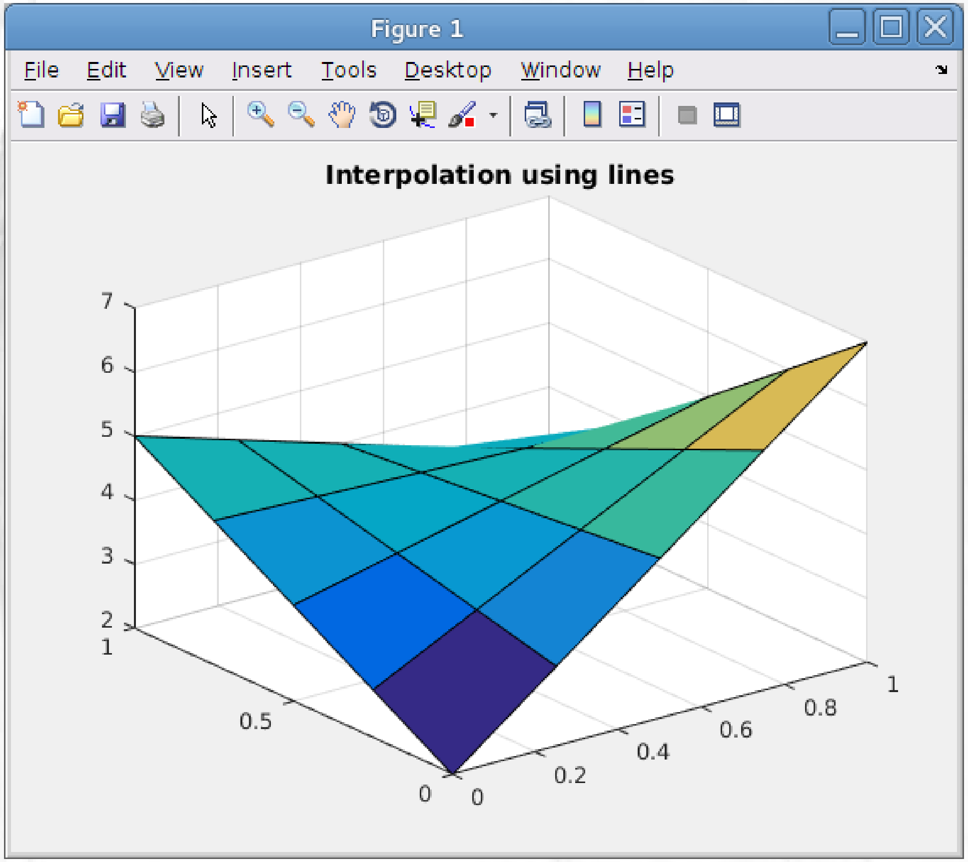
Inputs: [x, y], Coordinates of interpolation box.

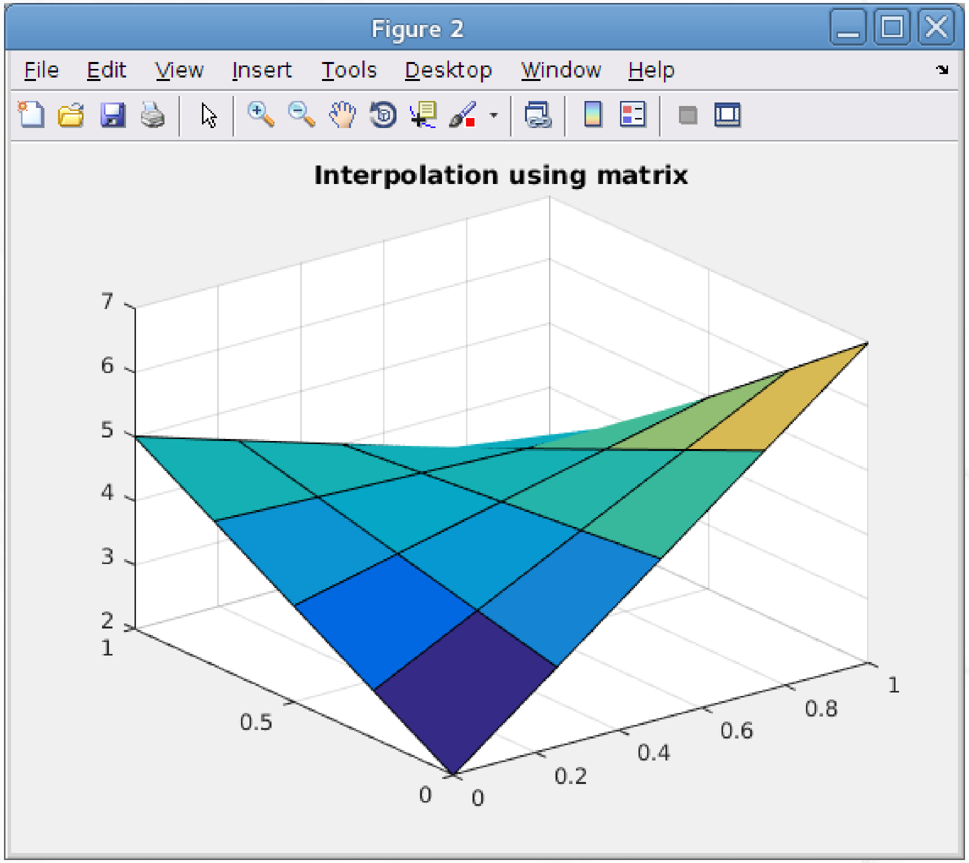
$$\begin{pmatrix} 1 & x_1 & y_1 & x_1 y_1 \\ 1 & x_2 & y_2 & x_2 y_2 \\ 1 & x_3 & y_3 & x_3 y_3 \\ 1 & x_4 & y_4 & x_4 y_4 \end{pmatrix} \begin{pmatrix} A \\ B \\ C \\ D \end{pmatrix} = \begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{pmatrix}$$

$$z(x, y) = A + B x + C y + D x y$$

- I claim: Both methods give the same answer.

“Proof by Matlab...”

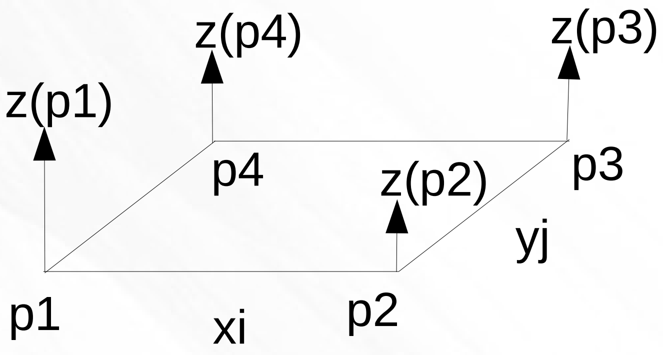





```
>> plot_interpolation
Norm of difference = 0.000000e+00
```

What you need to use this scheme for 2D surface interpolation

- Grid of (xi, yj) points.
- At each (xi, yj) point, value of fcn, zij.
- For each square, Aij, Bij, Cij, Dij coeffs.



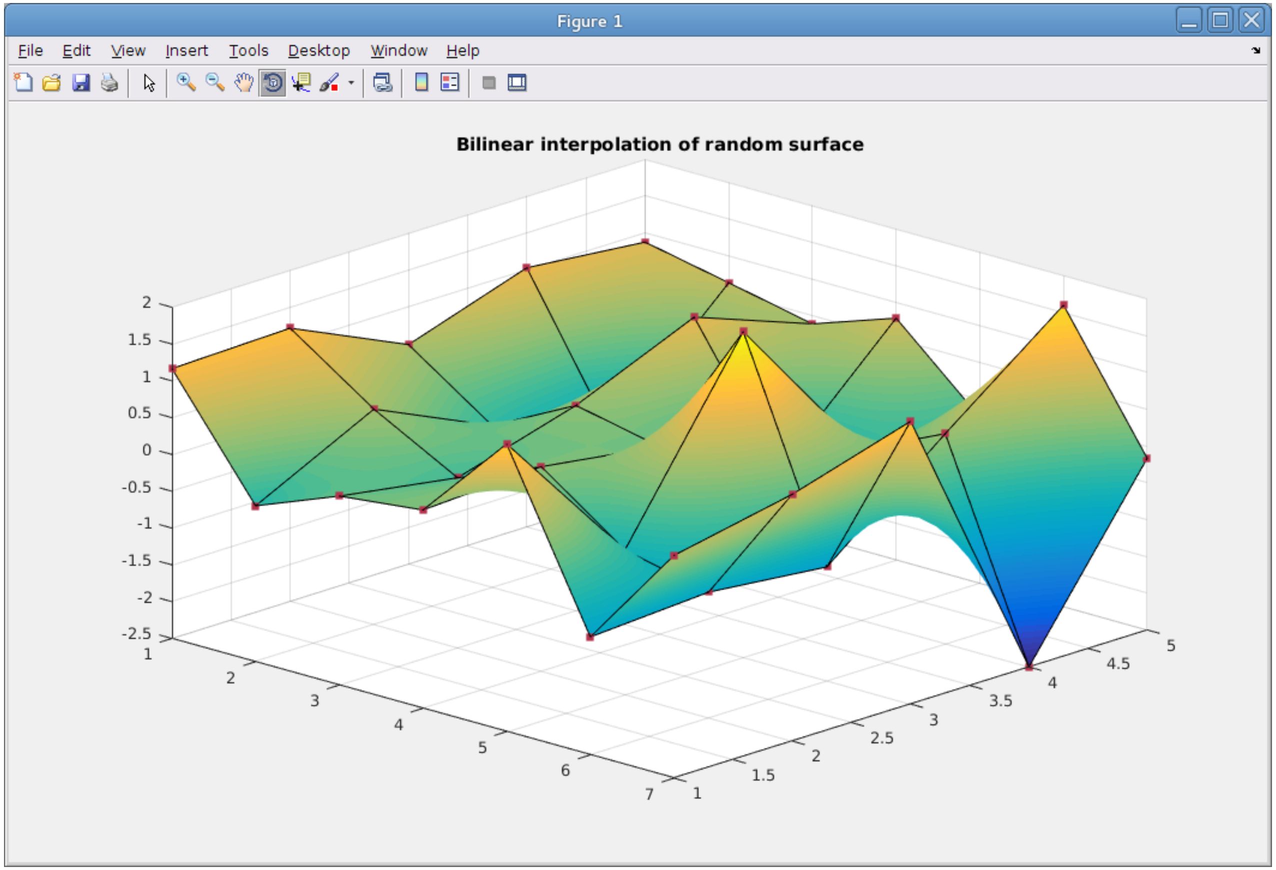
- Then, given arbitrary (x,y):
  - Find enclosing square.
  - Convert to local coords xi, yj.
  - Inside square, compute interpolated z using $z(x_i, y_j) = A + B\,x_i + C\,y_j + D\,x_i\,y_j$

Interpolation used for image processing

- Nearest neighbor
- Bilinear
- Bicubic
- NURBS (Non-uniform rational basis spline)
- Many others....

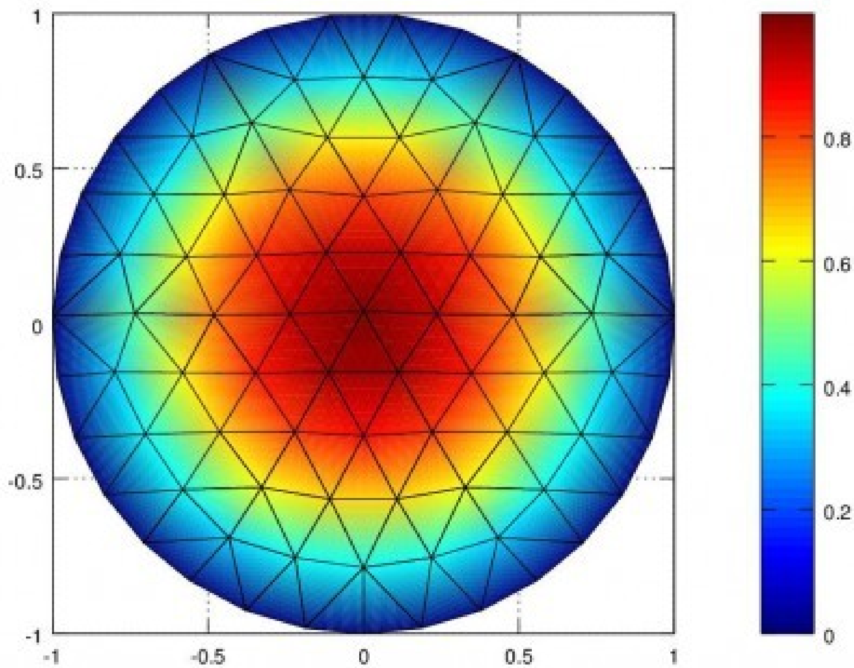
Interpolation and representing surfaces in 2 and 3D using triangular meshes

Surface reconstruction using bilinear interpolation on squares



- Requires evaluating polynomial inside each square.
- Surfaces have discontinuities at boundaries.

Another method: triangular mesh



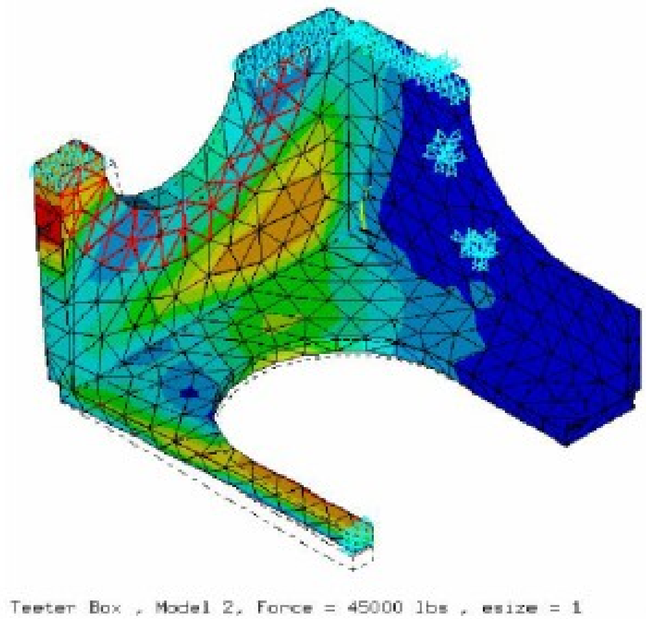
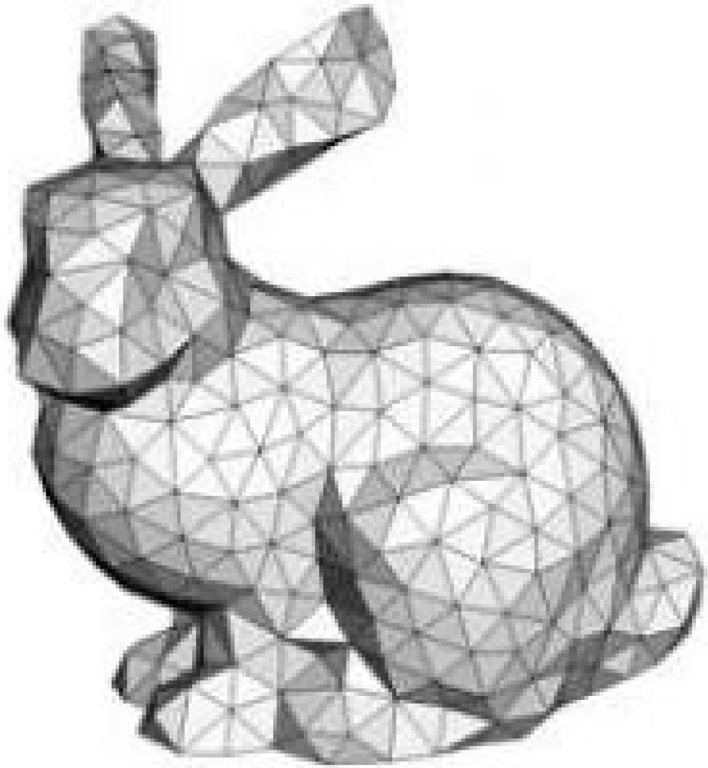
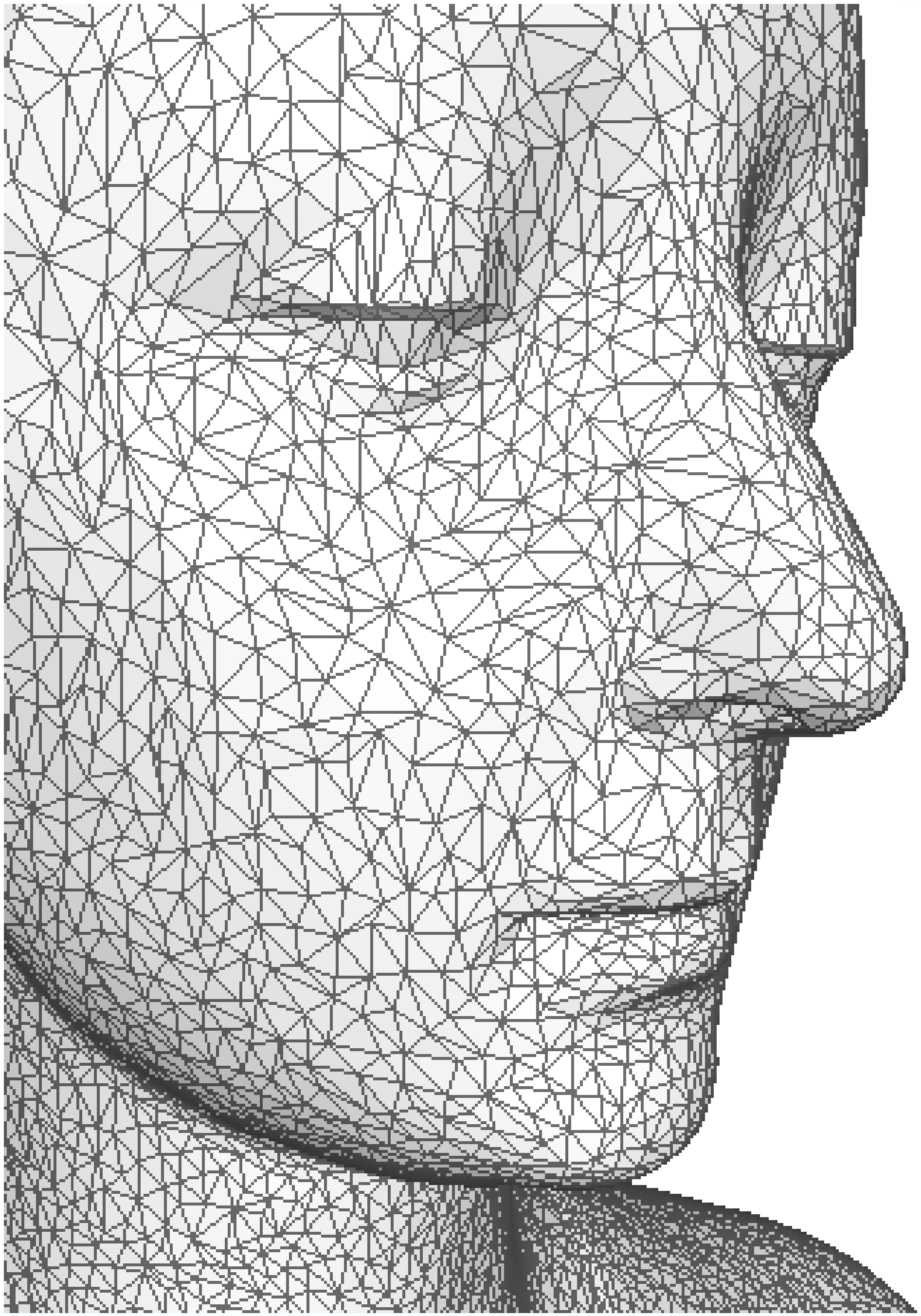


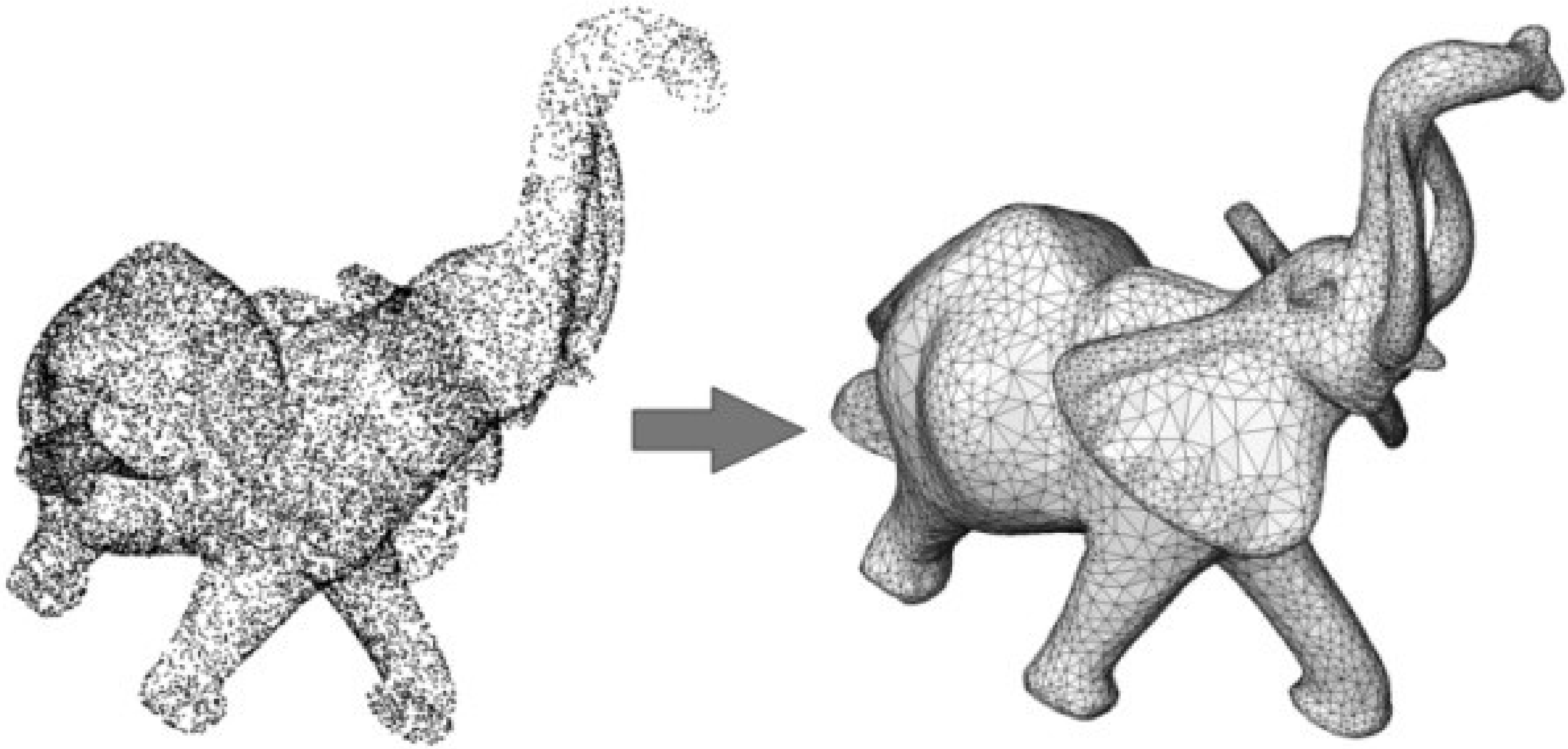
Figure 3
http://www.umass.edu/mie/labs/mda/fea/fealib/goldstein/PROJECT.html

- Cover domain with triangles.
- Triangles follow boundaries much better than simple squares.
- Triangles admit linear interpolation inside domain (squares require polynomial interpolation)

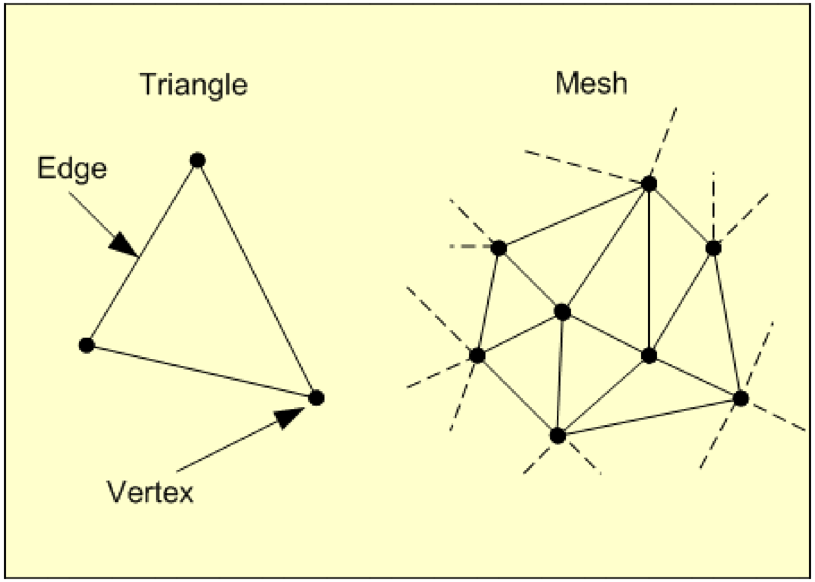
Workflow

- Create (or import) a point cloud.
- Create triangulation of point cloud. (This can actually be difficult, but ignore that for now.)
- Interpolate within triangles to create surfaces.

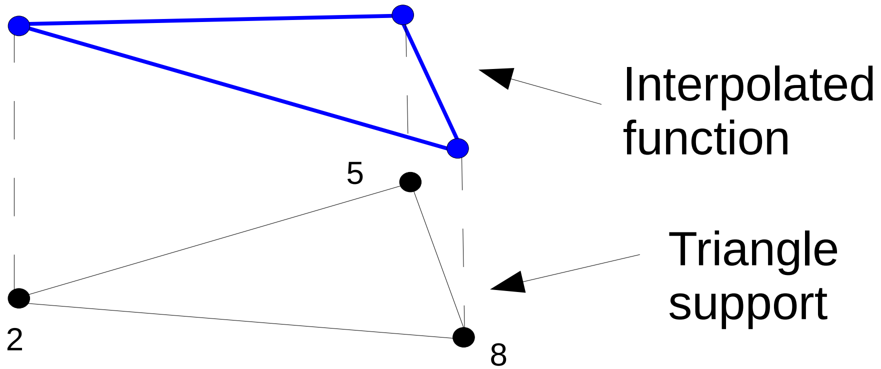


Using a triangular mesh to represent a surface

- Cover domain with triangles.
- Triangles have vertices (points).
- Triangles have edges (line segments).

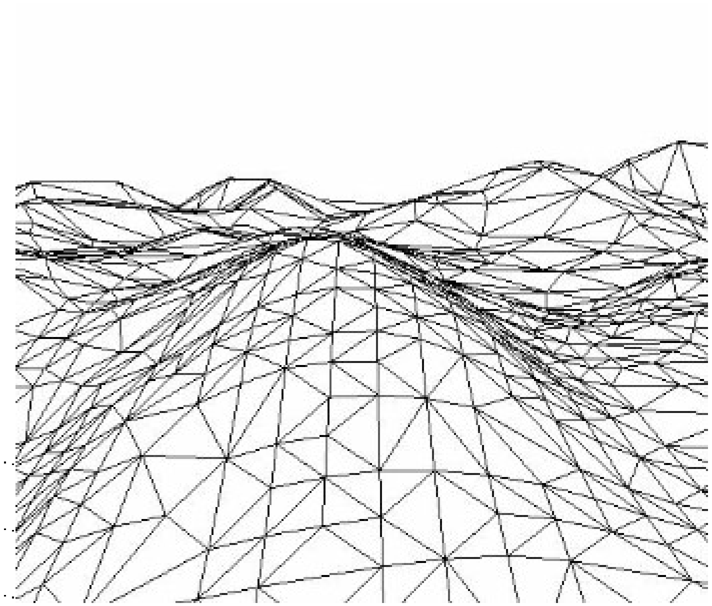


- Use interpolation to find value of function inside triangle. Approximate function by planar surface.

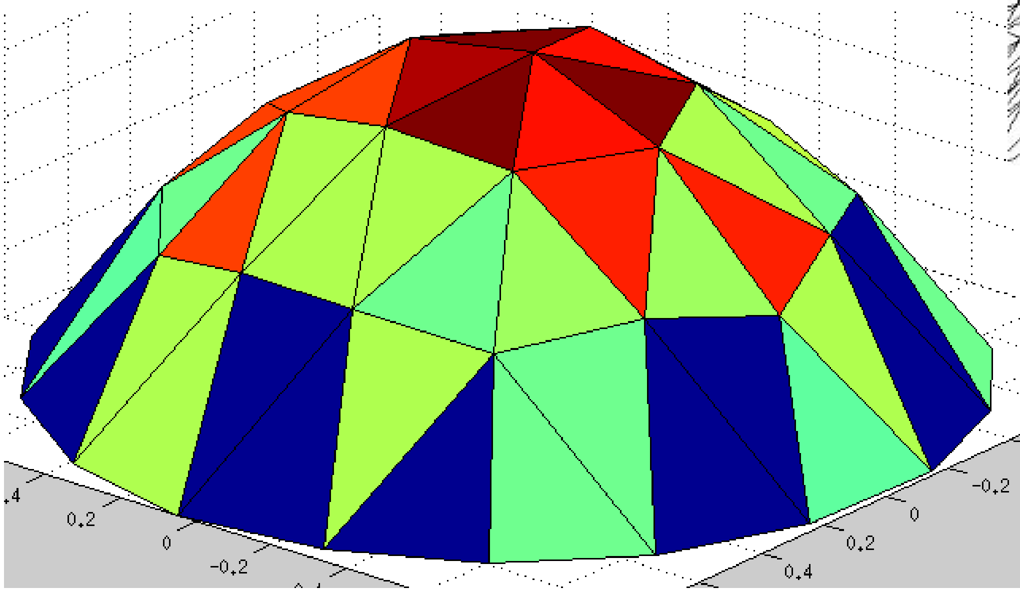


Representing a function using a triangular mesh – 2D linear interpolation

- Assemble individual triangles into function defined over domain



Arbitrary function

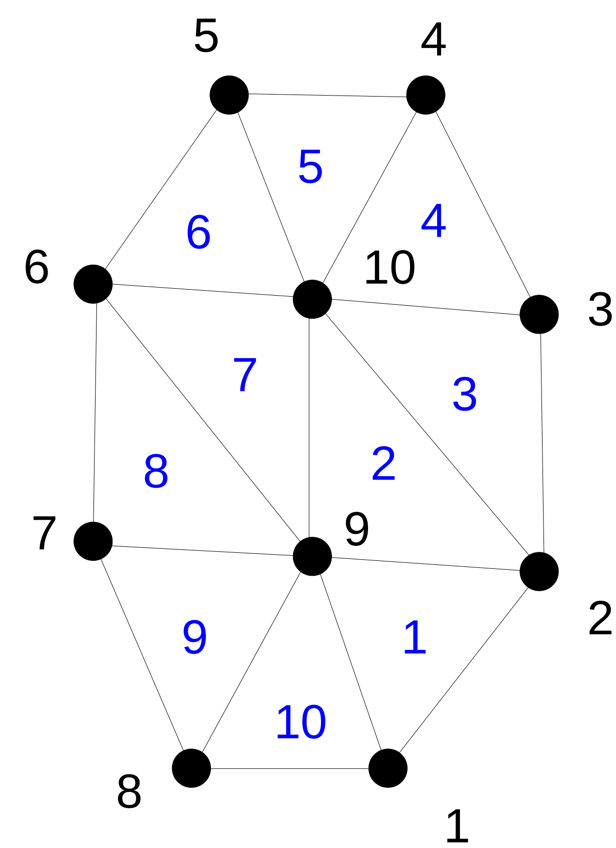


$$1-\left(x^2+y^2\right)$$

Representing a 2D mesh in Matlab

Tri index	Elements (triangles)			nIdx	Nodes (vertices)	
1	1	2	9	1	x1	y1
2	9	2	10	2	x2	y2
3	2	3	10	3	x3	y3
4	10	3	4	4	x4	y4
5	4	5	10	5	x5	y5
6	5	6	10	6	x6	y6
7	6	9	10	7	x7	y7
8	6	7	9	8	x8	y8
9	7	8	9	9	x9	y9
10	8	1	9	10	x10	y10

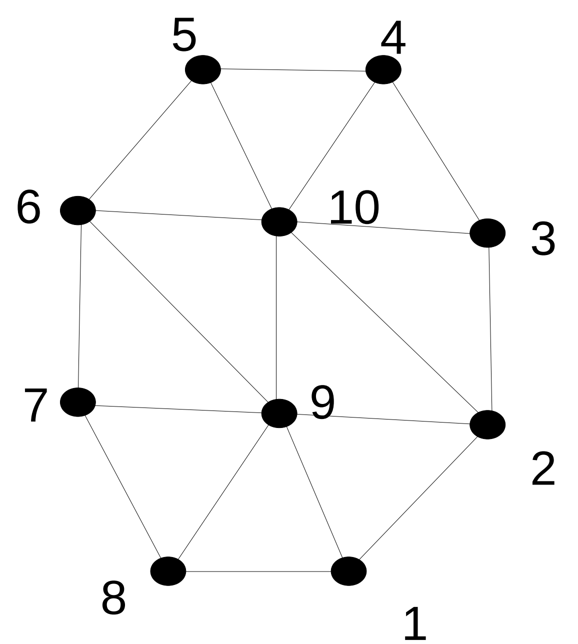
Note all nodes listed in CCW order.

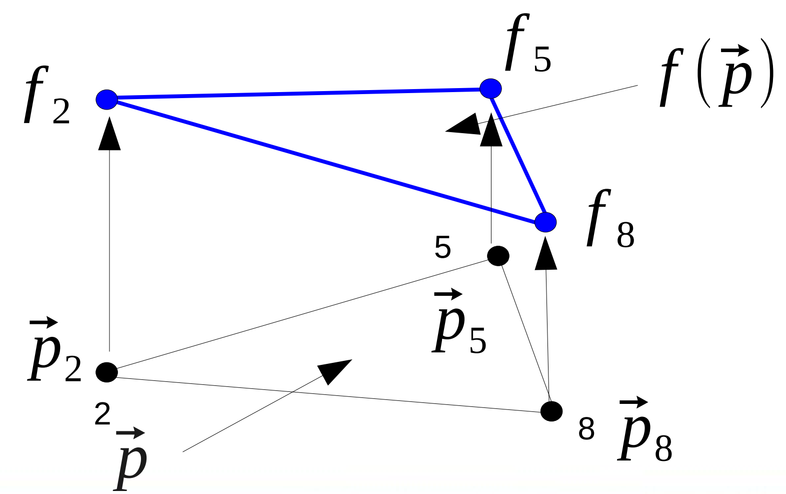


Nodes are black
Triangles (elements) are blue

Value of function defined at nodes

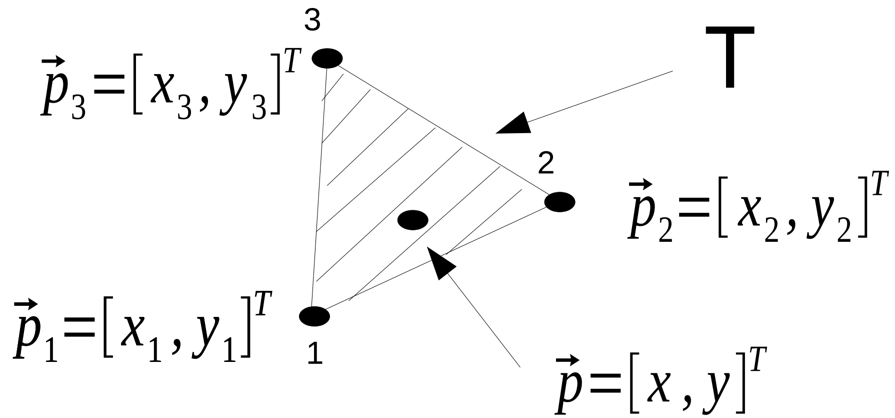
idx	Nodes (vertices)		Function values
1	x1	y1	f1
2	x2	y2	f2
3	x3	y3	f3
4	x4	y4	f4
5	x5	y5	f5
6	x6	y6	f6
7	x7	y7	f7
8	x8	y8	f8
9	x9	y9	f9
10	x10	y10	f10





Barycentric coordinates

- Given a triangle T with positions of three vertices known.
- Problem: Write an expression for an arbitrary (x, y) point inside the triangle.



- Barycentric coordinates:

$$\begin{pmatrix} x \\ y \end{pmatrix} = (1-\xi-\eta)\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \xi\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} + \eta\begin{pmatrix} x_3 \\ y_3 \end{pmatrix} \qquad \xi\in[0,1] \qquad \eta\in[0, 1-\xi]$$

- x, xi, y, yi known. ξ, η Unknown.

How to get unknowns?

- Rearrange

$$\xi\begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \end{pmatrix} + \eta\begin{pmatrix} x_3 - x_1 \\ y_3 - y_1 \end{pmatrix} = \begin{pmatrix} x - x_1 \\ y - y_1 \end{pmatrix}$$

- Or,

$$\begin{pmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{pmatrix}\begin{pmatrix} \xi \\ \eta \end{pmatrix} = \begin{pmatrix} x - x_1 \\ y - y_1 \end{pmatrix}$$

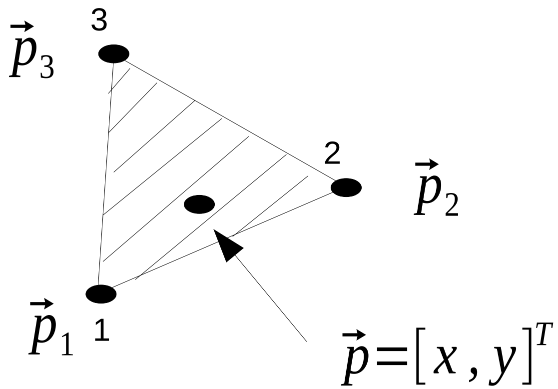
- Solve to get ξ, η

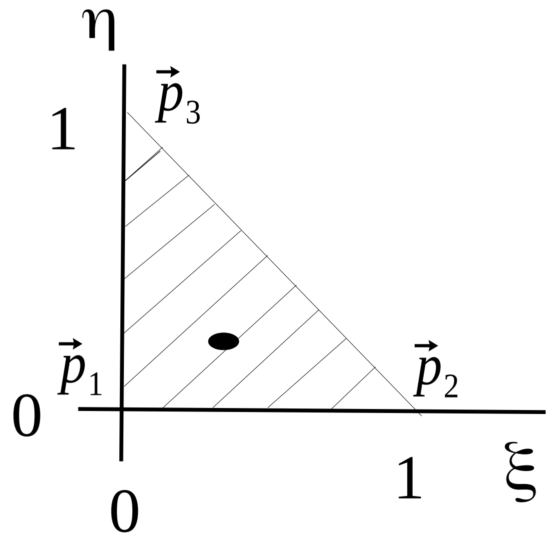
Barycentric coordinates

- Barycentric coordinates define mapping between points on an arbitrary triangle and the “standard triangle”

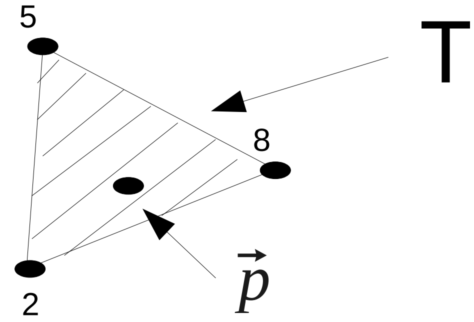
xi eta

$$\vec{p}=(1-\xi-\eta)\vec{p}_1+\xi\vec{p}_2+\eta\vec{p}_3 \qquad \xi\in[0,1] \qquad \eta\in[0,1-\xi]$$

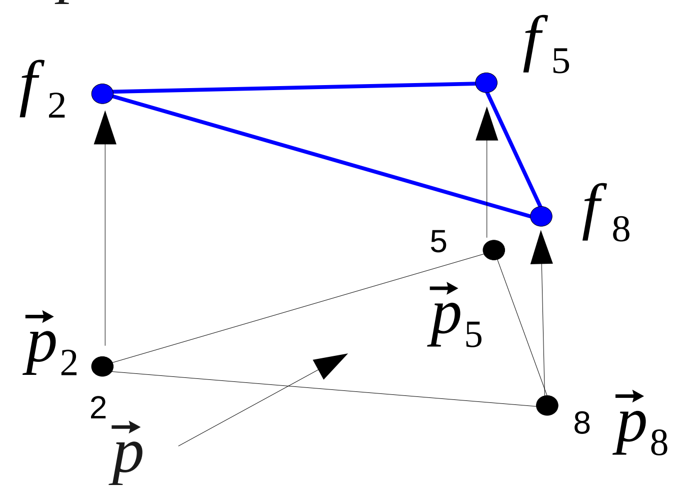




Interpolate planar surface over triangle



- What is value of f(x,y) at point *p*?



- We know function value at vertices
- Approximate f(x, y) as planar surface over T (analog to 1D linear interpolation).
- Barycentric interpolation on triangle

$$\vec{p}=(1-\xi-\eta)\vec{p}_2+\xi\vec{p}_8+\eta\vec{p}_5$$

Knowing x,y, get ξ,η

$$f(\vec{p})=(1-\xi-\eta)f_2+\xi f_8+\eta f_5$$

Use ξ,η to get interpolated fcn value

```
function interpolate_triangle()
  % This fcn fills in the area of a triangle.

  close all

  % N must be even number
  N = 14;
  eta = linspace(0, 1, N);
  xi = linspace(0, 1, N);

  p1 = [0; 0];
  p2 = [2; 1];
  p3 = [0.5; 3];

  % Fist make simple 2D plot of filled-in triangle
  figure(1)
  e = [p1, p2, p3, p1];
  plot(e(1,:), e(2,:), 'r')
  hold on

  for j = 1:N
    for i = 1:(N-j+1)
      p = (1-xi(i)-eta(j))*p1 + xi(i)*p2 + eta(j)*p3;
      plot(p(1), p(2), 'b*')
    end
  end
```

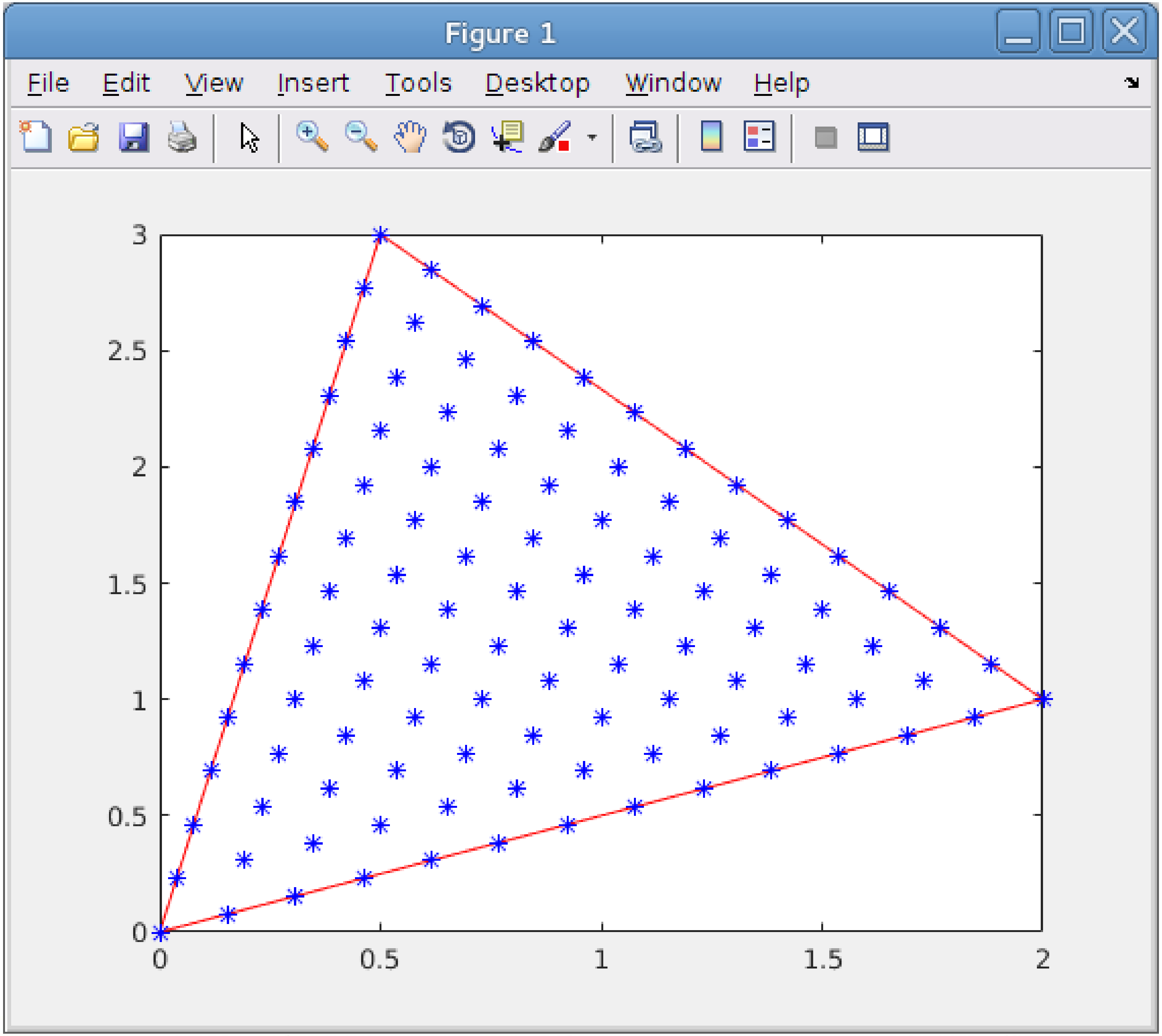


Figure 1
File Edit View Insert Tools Desktop Window Help
3
2.5
2
1.5
1
0.5
0
0
0.5
1
1.5
2

```
% Now demonstrate interpolation of surface
% Values of fcn at vertices
f1 = 2;
f2 = 2.5;
f3 = 1.2;

% Create vectors of [x, y, f] triplets
q1 = zeros(N*N/2, 1);
q2 = zeros(N*N/2, 1);
f = zeros(N*N/2, 1);
idx = 1;
for j = 1:N
  for i = 1:(N-j+1)
    q1(idx) = (1-xi(i)-eta(j))*p1(1) + xi(i)*p2(1) + eta(j)*p3(1);
    q2(idx) = (1-xi(i)-eta(j))*p1(2) + xi(i)*p2(2) + eta(j)*p3(2);
    f(idx) = (1-xi(i)-eta(j))*f1 + xi(i)*f2 + eta(j)*f3;
    idx = idx+1;
  end
end

fig2 = figure(2)
plot3(e(1,:), e(2,:), [0, 0, 0, 0], 'r')
hold on
set(fig2, 'units', 'normalized', 'position', [.2, .2, .6, .7])
plot3(e(1,:), e(2,:), [f1, f2, f3, f1], 'r')
plot3(q1, q2, f, 'bo')
```

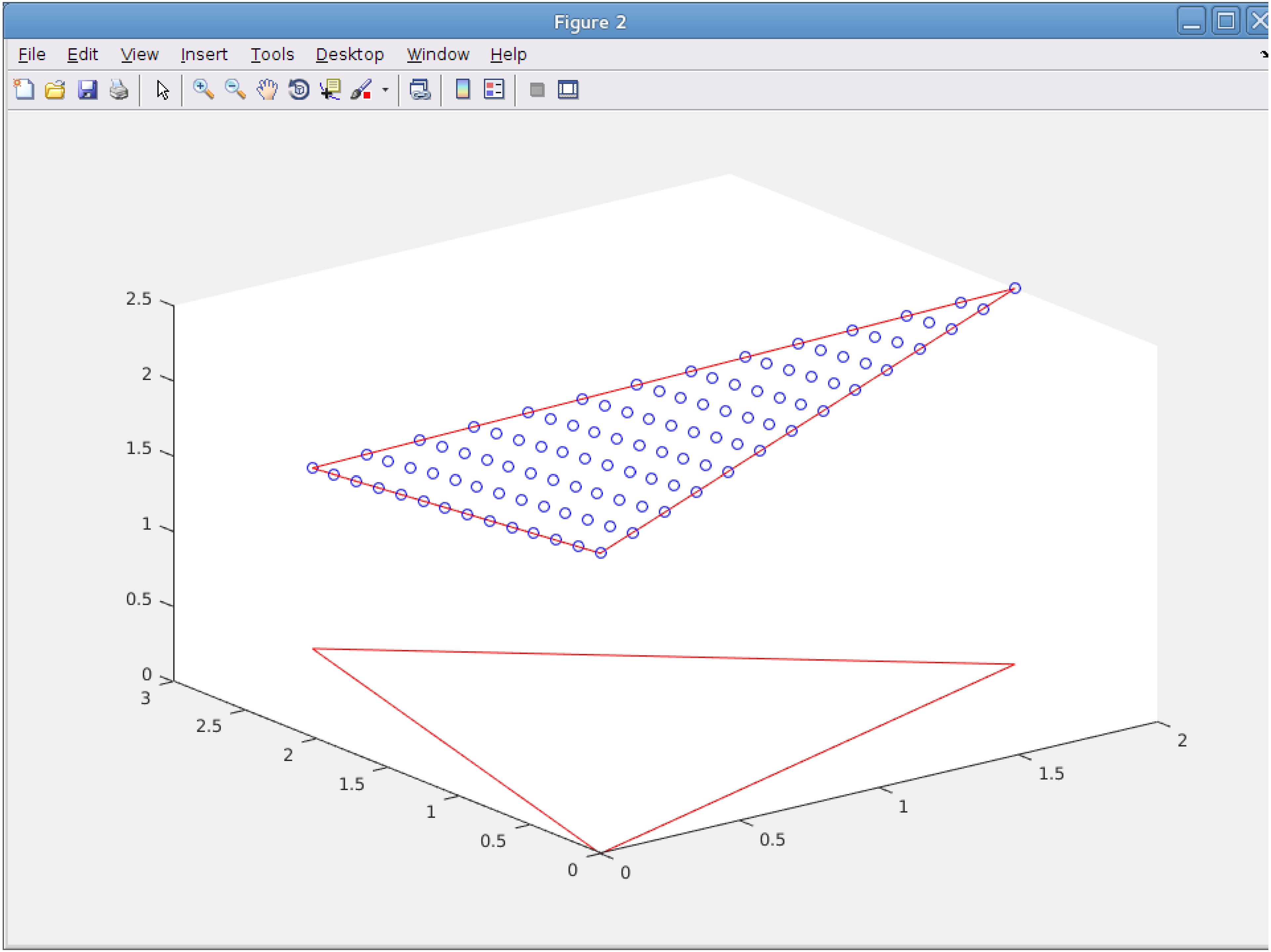
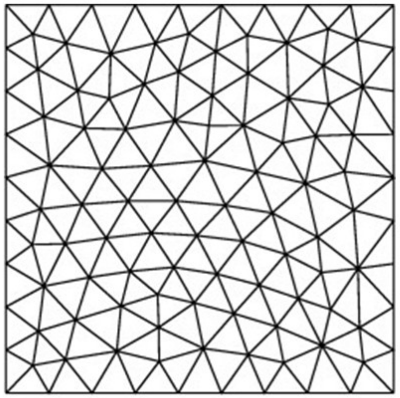
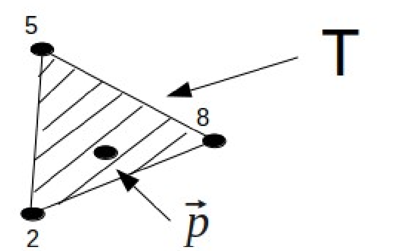


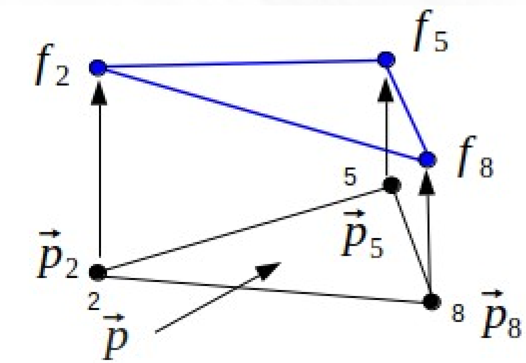
Figure 2
File
Edit
View
Insert
Tools
Desktop
Window
Help
2.5
2
1.5
1
0.5
0
3
2.5
2
1.5
1
0.5
0
0
0.5
1
1.5
2

What you need to use this scheme for 2D surface interpolation

- Mesh of triangles.
- At each vertex, value of fcn, zij.
- Then, given arbitrary (x,y):
  - Find enclosing triangle.
  - Compute local barycentric coords ξ, η
  - Use barycentric coords to do linear interpolation of surface.


Unstructured Mesh





Comparison of 2D methods

Bilinear interpolation on squares

- Interpolation via evaluating 2nd degree polynomial.
- Get polynomial coefficients for each square using 4x4 linear solve.
- Straightforward to figure out which square you are in.

Linear interpolation on triangles

- Interpolation via evaluating 1st degree polynomial (barycentric coordinates).
- Coefficients are interpolation points themselves.
- Must use a search and pointers to figure out which triangle you are in.

3 dimensions

- Interpolating in volumes uses tetrahedrons as elements.
- Problem of constructing a surface over a point cloud is more difficult.
  - Easy if you only want the convex hull
  - Difficult otherwise.
  - Name of problem: surface reconstruction

CGAL

CGAL 4.7 – Surface Reconstruction from Point Sets: User Manual – Mozilla Firefox

Mail - Brorso... | Class 8 – M... | about:config | DistMesh - A Si... | Chebyshev I... | Table of Fourier ... | UFLDL Tutorial -... | S0025-571... | Intel Data A... | CGAL 4.7 - Surf...

doc.cgal.org/latest/Surface_reconstruction_points_3/

barycentric interpolation triangle

cgal.org   Top   Getting Started   Organization of the Manual   Package Overview   Acknowledging CGAL   Search

CGAL 4.7 - Surface Reconstruction from Point Sets

- CGAL 4.7 - Surface Rec
  - User Manual
  - Reference Manual
  - Is Model Relationship
  - Bibliographic Referen
  - Class and Concept Li
  - Examples

User Manual

Authors
Pierre Alliez, Laurent Saboret, Gaël Guennebaud

1 Introduction
2 Common Reconstruction Pipeline
3 Poisson
3.1 Interface
3.2 Example
4 Contouring
5 Output
6 Case Studies
6.1 Ideal Conditions
6.2 Point Set
6.3 Contouring Parameters
6.4 Degraded Conditions
6.5 Sparse Sampling
6.6 Large Holes
6.7 Wrongly Oriented Normals
6.8 Noise and Outliers
6.9 Sharp Creases
7 Performances
7.1 Poisson Implicit Function
7.2 Contouring
7.3 Memory
7.4 Point Set Simplification
8 Design and Implementation History

1 Introduction

This CGAL component implements a surface reconstruction method which takes as input point sets with oriented normals and computes an implicit function. We assume that the input points contain no outliers and little noise. The output surface mesh is generated by extracting an isosurface of this function with the CGAL Surface Mesh Generator [4] or potentially with any other surface contouring algorithm.

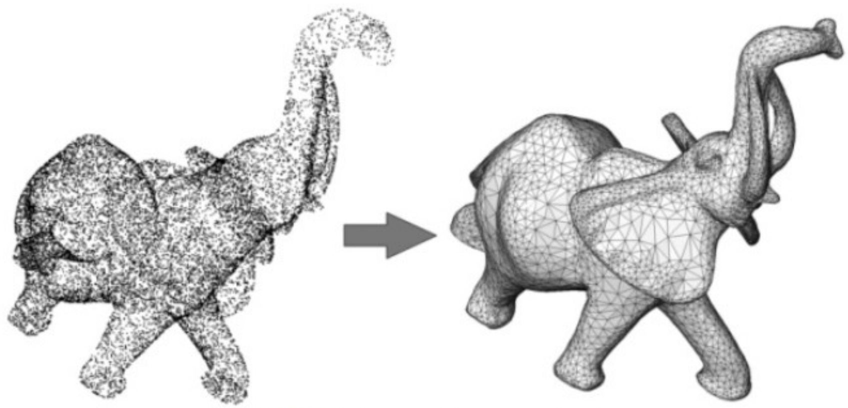
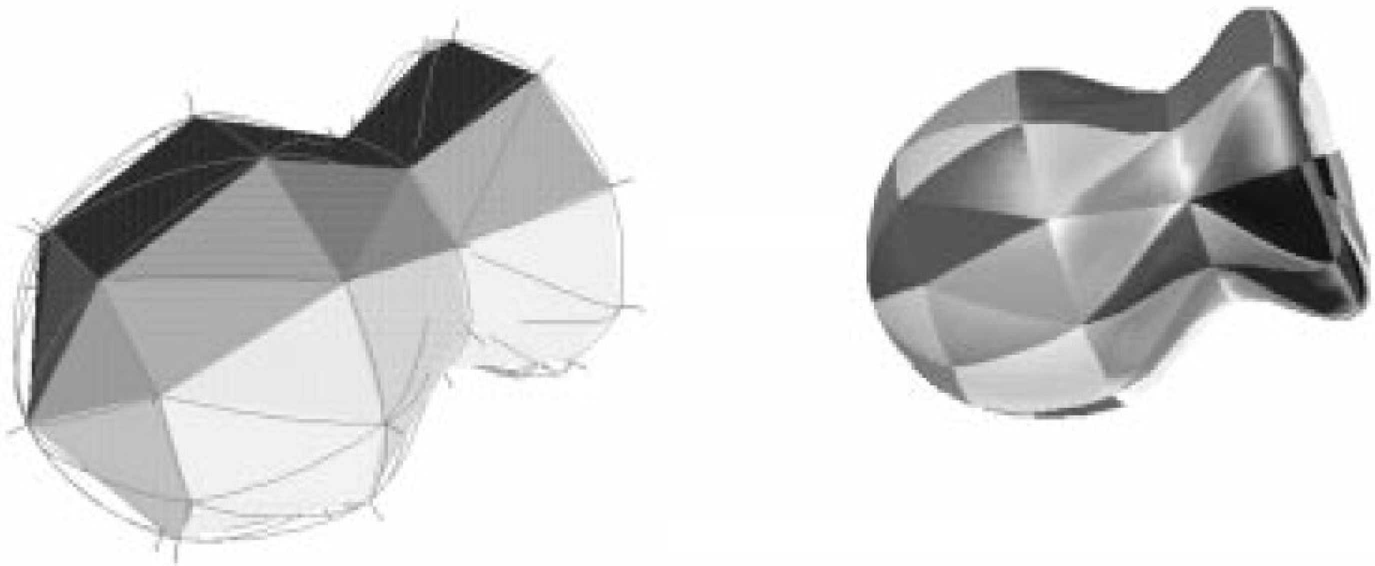


Figure 52.1 Poisson surface reconstruction.
Left: 17K points sampled on the statue of an elephant with a Minolta laser scanner. Right: reconstructed surface mesh.

More specifically, the core surface reconstruction algorithm consists of computing an implicit function which is an approximate indicator function of the inferred solid (Poisson Surface

Splines in higher dimensions



- Splines heavily used in computer graphics to generate display-ready curves and surfaces from underlying polygon mesh.
- GPUs are optimized to manipulate and solve small systems of linear equations. This makes them useful for displaying splines, triangulated surfaces, etc.

Major points from session

- 1D Splines
  - 3^{rd} degree polynomial typical.
- 2D bilinear interpolation on square domains
  - “Method of lines”
  - “Matrix method”
  - Useful for image processing.
- 2D linear interpolation on triangles.
  - Barycentric coordinates
  - Very common method for surface interpolation.