Math 541 - Numerical Analysis

Newton's Method in Higher Dimensions

Joseph M. Mahaffy,
⟨jmahaffy@mail.sdsu.edu⟩

Department of Mathematics and Statistics
Dynamical Systems Group
Computational Sciences Research Center
San Diego State University
San Diego, CA 92182-7720

http://jmahaffy.sdsu.edu

Spring 2018

Outline

U. S. Population Models

Source: Census Data

Below in a table of the U. S. population from census data

Year	Pop (M)	Year	Pop (M)	Year	Pop (M)
1790	3.929	1870	39.818	1950	150.697
1800	5.308	1880	50.189	1960	179.323
1810	7.240	1890	62.948	1970	203.302
1820	9.638	1900	76.212	1980	226.546
1830	12.866	1910	92.228	1990	248.710
1840	17.069	1920	106.022	2000	281.422
1850	23.192	1930	122.775	2010	308.746
1860	31.443	1940	132.165		

We'll use $t = 0$ as 1790

Malthusian Growth Model

The most basic population model is the **Malthusian growth model**,

$$P(t) = P_0 e^{rt}$$

We want the ***least squares best fit*** to the data:

$$P_0 e^{rt_i} = P_i, \quad i = 0, \dots, m,$$

which with **natural logarithms** this becomes a **linear model**:

$$\ln P_0 + rt_i = a_0 + a_1 t_i = \ln P_i.$$

A ***linear least squares fit*** to this problem gives the coefficients (a_0, a_1), where

$$P_0 = e^{a_0} \qquad \text{and} \qquad r = a_1.$$

Malthusian Growth Model

From the U. S. population data and fitting the $\ln(P)$, we find

$$a_0 = 1.843853 \qquad \text{and} \qquad a_1 = 0.0196234,$$

so the best fitting **Malthusian growth model** is

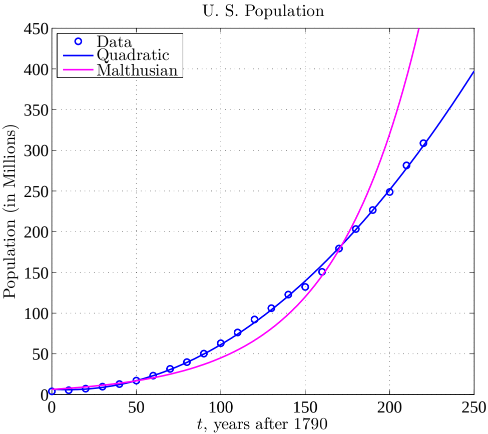
$$P(t) = 6.320851e^{0.0196234t}$$

The ***sum of square errors (SSE)*** for this model is

$$SSE = 48693.51.$$

This large error is largely caused by the bias of the logarithmic scale to over emphasize the early points.

Malthusian Growth Model - Graph

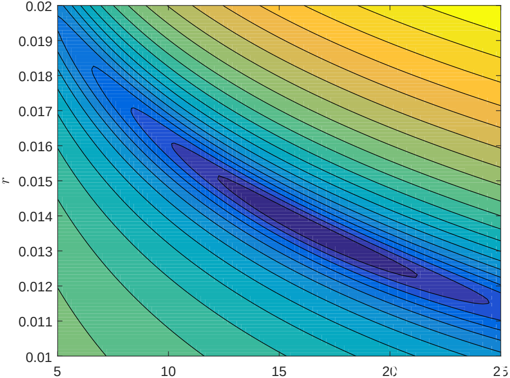


The graph shows the ***log fit*** parameters perform poorly for recent history. However, this was the **linear fit** to the **logarithm of the population data.**

Sum of Square Errors

The **sum of square errors** satisfies

$$E(P_0, r) = \sum_{i=0}^{n} \left(P_0 e^{r t_i} - P_i\right)^2$$



Sum of Square Errors

The contour plot is produced by the following:

```
% Create a contour of population lst sq
load 'uspop';
p0 = linspace(5,25,100);
r  = linspace(0.01,0.02,100);
[X,Y] = meshgrid(p0,r);
Z = zeros(100);
for i=1:100
    for j=1:100
        Z(j,i) = mallstsqc(p0(i),r(j),t,pop);
    end
end

figure(1)
contourf(X,Y,log(Z),20)
xlabel('$P_0$','interpreter','latex')
ylabel('$r$','interpreter','latex')
```

Sum of Square Errors

The ***sum of square errors*** function is given by

```
function LS = mallstsqc(p0,r,t,y)
%Least Squares sum of square errors to Malthusian ...
    growth model
LS = sum((p0*exp(r*t)-y).^2);
end
```

The contour plot shows a classic **banana-shaped** curve with a long stretched **minimum**

Graphically, the **minimum** is clearly far from the parameters found by the **linear fit** to the **logarithms** of the data

However, this graph shows the **sum of square errors** for the **nonlinear Malthusian growth model**

Method of Steepest Descent

The **sum of square errors** satisfies

$$E(P_0, r) = \sum_{i=0}^{n} \left(P_0 e^{rt_i} - P_i\right)^2$$

is a function of **two** variables

Recall from Calculus that $-\nabla E(P_0, r)$ (opposite the **gradient**) gives the direction of **steepest descent**, perpendicular to the level curve contours

- **Method of Steepest Descent**
  - Gradient gives the directional derivative
  - Follow path opposite ($x_n = [P_0^{(n)}, r^{(n)}]$)

    $$x_{n+1} = x_n - \alpha \nabla E(x_n)$$

  - Find α such that $E(x_{n+1})$ is much smaller than $E(x_n)$
  - "Banana curves" are notoriously slow for this method

Nelder-Mead Simplex Method

Define the **sum of square errors** with the **two** parameters

$$E(P_0, r) = \sum_{i=0}^{n} \left(P_0 e^{rt_i} - P_i\right)^2$$

Least Squares lecture showed MatLab's `fminsearch`

```
[p,err]=fminsearch(@mallstsq,p0,[],t,pop)
```

giving the best model as

$$P(t) = 16.345612e^{0.0136284t},$$

- **Nelder-Mead Simplex Method**
  - Requires a "good" initial guess
  - Uses triangular simplexes
  - Algorithm searches space with **three** function evaluations. (No derivatives!)
  - Computes centroid, then improves one of the three points
  - Details for a future class, **Math 693A**?

Minimum – Sum of Square Errors

The graph of the **sum of square errors** shows a distinct minimum

The function of **two variables** is

$$E(P_0, r) = \sum_{i=0}^{n} \left(P_0 e^{rt_i} - P_i\right)^2$$

From Calculus, a necessary condition for a minimum is

$$\nabla E(P_0, r) = \tilde{\mathbf{0}}$$

Taking partial derivatives gives

$$\begin{aligned} \frac{\partial E}{\partial P_0} &= 2\sum_{i=0}^{n} \left(P_0 e^{rt_i} - P_i\right) e^{rt_i} \\ \frac{\partial E}{\partial r} &= 2\sum_{i=0}^{n} \left(P_0 e^{rt_i} - P_i\right) P_0 t_i e^{rt_i} \end{aligned}$$

Nonlinear Least Squares - Malthusian Growth

The minimum occurs when these partial derivatives are **zero**

This requires solving two nonlinear equations for the parameters, P_0 and r

$$\begin{aligned}\sum_{i=0}^{n}\left(P_0 e^{rt_i} - P_i\right) e^{rt_i} &= 0\\ \sum_{i=0}^{n}\left(P_0 e^{rt_i} - P_i\right) P_0 t_i e^{rt_i} &= 0\end{aligned}$$

Note that the first equation could be solved for P_0 easily

This could be substituted into the second equation, and the resulting nonlinear equation could be solved by Newton's method or one of our other routine for solving $f(x) = 0$

Minimization Problem

Consider a function $f : \mathbb{R}^n \to \mathbb{R}$

Our problem is to minimize $f(\tilde{\mathbf{x}})$

$$\min_{\tilde{\mathbf{x}} \in \mathbb{R}^n} f(\tilde{\mathbf{x}}),$$

where

- $f(\tilde{\mathbf{x}})$ is the **objective function**
- $\tilde{\mathbf{x}}$ is the vector of variables or parameters

In our example above, the function f is the ***sum of square errors***, and the vector $\tilde{\mathbf{x}} = [P_0, r]$

Minimization Problem

If $f(\tilde{\mathbf{x}}) \in \mathbb{R}$ is **differentiable** we can recognize a **minimum** by looking at the first and second derivatives:

- The **gradient**: $\nabla f(\tilde{\mathbf{x}}) \in \mathbb{R}^n$
- The **Hessian**: $\nabla^2 f(\tilde{\mathbf{x}}) \in \mathbb{R}^{n \times n}$

Once again **Taylor's Theorem** (multi-dimensional) plays a key role

Theorem (First Order Necessary Condition)

If $\tilde{\mathbf{x}}^*$ is a ***local minimum*** *and f is* ***continuously differentiable*** *in an open neighborhood of $\tilde{\mathbf{x}}^*$, then*

$$\nabla f(\tilde{\mathbf{x}}^*) = \tilde{\mathbf{0}}.$$

Local Minimum

Definition (Positive Definite Matrix)

An $n \times n$ matrix H is **positive definite** if and only if for all $\tilde{\mathbf{x}} \neq 0$,

$$\tilde{\mathbf{x}}^T H \tilde{\mathbf{x}} = \sum_{i=1}^{n} \sum_{j=1}^{n} h_{ij} x_i x_j > 0.$$

Theorem (Second-Order Sufficient Conditions)

Suppose that $\nabla^2 f$ is ***continuous*** *in an open neighborhood of $\tilde{\mathbf{x}}^*$ and that $\nabla f(\tilde{\mathbf{x}}^*) = \tilde{\mathbf{0}}$ and $\nabla^2 f(\tilde{\mathbf{x}}^*)$ is* ***positive definite****. Then $\tilde{\mathbf{x}}^*$ is a* ***strict local minimum*** *of f.*

Note: This is similar in 1-D of the derivative equal to zero and the second derivative being positive for a ***local minimum***

Minimization Problem

We generally do not have a global picture of $f(\tilde{\mathbf{x}})$, so we concentrate on the local problem

Relying on our techniques from Calculus and the theorems above, we seek to solve

$$\nabla f(\tilde{\mathbf{x}}) = \tilde{\mathbf{0}}.$$

We will concentrate on the case where there is a ***local minimum***, so the **Hessian** is ***positive definite***. (**Math 693A** will go into much more detail.)

We limit our discussion here to **Newton's Method**, which is a **line search** technique

Line Search Method

Line search methods reduce the n-dimensional ***minimization problem***

$$\min_{\tilde{\mathbf{x}} \in \mathbb{R}^n} f(\tilde{\mathbf{x}}),$$

with the one-dimensional problem

$$\min_{\tilde{\mathbf{x}} \in \mathbb{R}^n} f(\tilde{\mathbf{x}}_k + \alpha \tilde{\mathbf{p}}_k),$$

where $\tilde{\mathbf{p}}_k$ is a chosen ***search direction*** and α is the factor for how far we search in that direction.

Choice of $\tilde{\mathbf{p}}_k$ is critical to the rate of progress toward the ***local minimum***

The intuitive ***steepest descent*** actually gives a slow scheme

Taylor Theorem

The **fundamental** building block for our ***minimization problem*** is once again **Taylor's Theorem** in higher dimensions

Theorem (Taylor's Theorem)

For some $t \in (0,1)$, *we have*

$$f(\tilde{\mathbf{x}} + \tilde{\mathbf{p}}) = f(\tilde{\mathbf{x}}) + \tilde{\mathbf{p}}^T \underbrace{\nabla f(\tilde{\mathbf{x}})}_{gradient} + \frac{1}{2}\tilde{\mathbf{p}}^T \underbrace{\left[\nabla^2 f(\tilde{\mathbf{x}} + t\tilde{\mathbf{p}})\right]}_{Hessian} \tilde{\mathbf{p}}.$$

The ***local minimum*** occurs when we find $\tilde{\mathbf{x}}^*$, such that $\nabla f(\tilde{\mathbf{x}}^*) = \tilde{\mathbf{0}}$ and $\nabla^2 f(\tilde{\mathbf{x}}^*)$ is positive definite

Newton Direction

If f is sufficiently smooth and the Hessian is positive definite, we can select $\tilde{\mathbf{p}}_k$ to be the **Newton direction**.

The second order Taylor expansion gives:

$$f(\tilde{\mathbf{x}} + \tilde{\mathbf{p}}) \approx f(\tilde{\mathbf{x}}) + \tilde{\mathbf{p}}^T \nabla f(\tilde{\mathbf{x}}) + \frac{1}{2} \tilde{\mathbf{p}}^T \left[\nabla^2 f(\tilde{\mathbf{x}}) \right] \tilde{\mathbf{p}}$$

The minimum of the rhs is obtained by computing the derivative with respect to $\tilde{\mathbf{p}}$ and setting the result to zero, so

$$\nabla f(\tilde{\mathbf{x}}) + \left[\nabla^2 f(\tilde{\mathbf{x}}) \right] \tilde{\mathbf{p}} = \tilde{\mathbf{0}},$$

which gives the **Newton direction**

$$\tilde{\mathbf{p}}^N = - \left[\nabla^2 f(\tilde{\mathbf{x}}) \right]^{-1} \nabla f(\tilde{\mathbf{x}}).$$

Newton's Method or Algorithm

Given an initial $\tilde{\mathbf{x}}_0$, **Newton's Method** for finding a ***minimum*** for $f(\tilde{\mathbf{x}})$ is iteratively given by

$$\tilde{\mathbf{x}}_{n+1} = \tilde{\mathbf{x}}_n - \left[\nabla^2 f(\tilde{\mathbf{x}}_n)\right]^{-1} \nabla f(\tilde{\mathbf{x}}_n).$$

More generally, let $\tilde{\mathbf{g}}(\tilde{\mathbf{x}}) : \mathbb{R}^n \to \mathbb{R}^n$, then **Newton's Method** can be used to find a solution of

$$\tilde{\mathbf{g}}(\tilde{\mathbf{x}}) = \tilde{\mathbf{0}} \qquad \text{given initial} \quad \tilde{\mathbf{x}}_0.$$

The **Newton iteration scheme** satisfies

$$\tilde{\mathbf{x}}_{n+1} = \tilde{\mathbf{x}}_n - J^{-1}(\tilde{\mathbf{x}}_n)\tilde{\mathbf{g}}(\tilde{\mathbf{x}}_n),$$

where $J(\tilde{\mathbf{x}})$ is the **Jacobian matrix** for $\tilde{\mathbf{g}}(\tilde{\mathbf{x}})$

Newton's Method for $\mathbb{R}^3$

Consider $\tilde{\mathbf{g}}(\tilde{\mathbf{x}}) : \mathbb{R}^3 \to \mathbb{R}^3$, where

$$\tilde{\mathbf{g}}(\tilde{\mathbf{x}}) = \begin{pmatrix} g_1(\tilde{\mathbf{x}}) \\ g_2(\tilde{\mathbf{x}}) \\ g_3(\tilde{\mathbf{x}}) \end{pmatrix} \qquad \text{and} \qquad \tilde{\mathbf{x}} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

The **Jacobian matrix** is given by

$$J(\tilde{\mathbf{x}}) = \begin{pmatrix} \frac{\partial g_1(\tilde{\mathbf{x}})}{\partial x_1} & \frac{\partial g_1(\tilde{\mathbf{x}})}{\partial x_2} & \frac{\partial g_1(\tilde{\mathbf{x}})}{\partial x_3} \\ \frac{\partial g_2(\tilde{\mathbf{x}})}{\partial x_1} & \frac{\partial g_2(\tilde{\mathbf{x}})}{\partial x_2} & \frac{\partial g_2(\tilde{\mathbf{x}})}{\partial x_3} \\ \frac{\partial g_3(\tilde{\mathbf{x}})}{\partial x_1} & \frac{\partial g_3(\tilde{\mathbf{x}})}{\partial x_2} & \frac{\partial g_3(\tilde{\mathbf{x}})}{\partial x_3} \end{pmatrix}$$

Then given an initial $\tilde{\mathbf{x}}_0$, **Newton's Method** can be used to find an approximate solution of $\tilde{\mathbf{g}}(\tilde{\mathbf{x}}) = \tilde{\mathbf{0}}$, where

$$\tilde{\mathbf{x}}_{n+1} = \tilde{\mathbf{x}}_n - J^{-1}(\tilde{\mathbf{x}}_n)\tilde{\mathbf{g}}(\tilde{\mathbf{x}}_n)$$

$\mathbb{R}^2$ Example of Newton's Method 1

Suppose we want to find the intersection of the 2D curves

$$x_1^2 + x_2^2 = 4 \qquad \text{and} \qquad x_1 x_2 = 1$$

(This could be easily solved in 1D, then back substituted.)

We create a $\tilde{\mathbf{g}}(x_1, x_2)$ with $\tilde{\mathbf{g}}(x_1, x_2) = \tilde{\mathbf{0}}$

$$\tilde{\mathbf{g}}(x_1, x_2) = \begin{pmatrix} x_1^2 + x_2^2 - 4 \\ x_1 x_2 - 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

The **Jacobian matrix** is given by

$$J(\tilde{\mathbf{x}}) = \begin{pmatrix} 2x_1 & 2x_2 \\ x_2 & x_1 \end{pmatrix} \qquad \text{so} \qquad J^{-1}(\tilde{\mathbf{x}}) = \begin{pmatrix} \frac{x_1}{2(x_1^2 - x_2^2)} & -\frac{x_2}{(x_1^2 - x_2^2)} \\ -\frac{x_2}{2(x_1^2 - x_2^2)} & \frac{x_1}{(x_1^2 - x_2^2)} \end{pmatrix}$$

Given an initial $\tilde{\mathbf{x}}_0$, **Newton's Method** satisfies

$$\tilde{\mathbf{x}}_{n+1} = \tilde{\mathbf{x}}_n - J^{-1}(\tilde{\mathbf{x}}_n)\tilde{\mathbf{g}}(\tilde{\mathbf{x}}_n)$$

$\mathbb{R}^2$ Example of Newton's Method 2

Given an initial $\tilde{\mathbf{x}}_0$, a MatLab version of **Newton's Method** is

```
function xn = newton2(x0,tol,N)
% 2D Newton's Method
xn = x0; err = 1; k = 1;
while ((err > tol)&&(k < N))
    x1 = xn;
    xn = x1 - J2(x1)\g2(x1)
    err = abs(norm(x1)-norm(xn))
    k = k+1;
end
end
```

$\mathbb{R}^2$ Example of Newton's Method 3

The MatLab function and Jacobian are

```
function y = g2(x)
% 2D function
y = [x(1,1)^2 + x(2,1)^2 - 4; x(1,1)*x(2,1) - 1];
end
```

```
function A = J2(x)
% Jacobian matrix
A = [2*x(1,1), 2*x(2,1); x(2,1) ,x(1,1)];
end
```

$\mathbb{R}^2$ Example of Newton's Method 4

Results of the program are listed in the Table below:

x_1	x_2	Norm Err
2	0	-
2	0.5	0.061553
1.933333	0.516667	0.060373
1.931853	0.517637	0.0011794
1.931851	0.517638	$7.8345E-07$
1.931851	0.517638	$5.6444E-13$

Note: The norm error is in the Cauchy sense comparing successive iterations.

It can be shown that this **Newton's Method** converges **quadratically**

Symmetry shows there are 4 solutions

Newton's Method and Population Model

The Newton scheme derived above can be applied to the ***least squares function*** for our **Malthusian growth model**

However, the large number of data points and the exponentials result in very large gradients, particularly in the r direction

This results in serious numerical instabilities of our **Newton's method**

Below is the **MatLab function** for the gradient:

```
function y = gpop(x,t,pop)
% 2D function
y1 = sum((x(1,1)*exp(x(2,1)*t)-pop).*exp(x(2,1)*t));
y2 = sum((x(1,1)*exp(x(2,1)*t)-pop).*(x(1,1)*t)...
     .*exp(x(2,1)*t));
y = [y1;y2];
end
```

Newton's Method and Population Model

We readily compute the Jacobian in **MatLab**:

```
function A = Jpop(x,t,pop)
% Jacobian matrix
a11 = sum(exp(2*x(2,1)*t));
a12 = sum(2*x(1,1).*t.*exp(2*x(2,1)*t) - ...
    pop.*t.*exp(x(2,1)*t));
a21 = a12;
a22 = sum(2*x(1,1)^2*(t.^2).*exp(2*x(2,1)*t) - ...
    x(1,1).*(t.^2).*exp(x(2,1)*t).*pop);
A = [a11,a12;a21,a22];
end
```

Newton's Method and Population Model

Newton's method becomes in **MatLab**:

```
function xn = newton_pop(x0,t,pop,tol,N)
% 2D Newton's Method
xn = x0; err = 1; k = 1;
while ((err > tol)&&(k < N))
    x1 = xn;
    xn = x1 - Jpop(x1,t,pop)\gpop(x1,t,pop)
    err = abs(norm(x1)-norm(xn))
    k = k+1
end
end
```

Newton's Method and Population Model

Recall that the ***linear least squares best fit*** to the logarithm of the data was a long distance from the observed minimum (Contour plot).

We attempt starting **Newton's method** at this point $(P_0, r) = (6.32, 0.0196)$, and the Table below is generated with n the number of iterations

n	P_0	r	err
1	5.8604	0.01906	0.4596
2	1.8088	0.02213	4.0515
3	2.1092	0.02814	0.3004
4	2.0917	0.02636	0.01753
5	2.0342	0.02505	0.05743
10	0.2568	0.007638	0.007617
12	0.2854	-0.003745	0.01724

Newton's Method and Population Model

The previous slide shows **Newton's method** diverging.

From the contour map a good starting point for **Newton's method** is $(P_0, r) = (16, 0.014)$ with the Table below showing the iterations.

n	P_0	r	err
1	15.0906	0.0140456	0.9094
2	16.3185	0.0136219	1.2279
3	16.3437	0.0136291	0.02517
4	16.3456	0.0136284	0.001898
5	16.3456	0.0136284	3.337E-07

This Table shows **Newton's method** converging quadratically

Nelder-Mead Method

A number of times this semester, we have employed the **MatLab minimization algorithm** `fminsearch`.

This algorithm employs the **Nelder-Mead method**[1], which is a heuristic technique based on searching parameter space with ***simplexes***.

- This simplex method finds a ***local minimum*** of a function of several variables.
- If $f(x,y)$ is function of two variables, the pattern starts with an initial triangle.
- The largest vertex is rejected and replaced with another.
- The process generates a sequence of triangles with vertices having smaller functional values.
- The size of the triangles eventually reduces to where it is sufficiently close to the ***local minimum***.

[1] Nelder, John A., R. Mead (1965). A simplex method for function minimization. *Computer Journal* **7**: 308313.

Nelder-Mead Method

The **Nelder-Mead algorithm** is developed using a ***simplex***, which is a generalized triangle in N dimensions, and it follows an effective and computationally compact scheme.

For clarity, the **Nelder-Mead algorithm** is shown for $f(x,y)$, a function of two variables[2].

Let $f(x,y)$ be a function to be minimized, and select an initial triangle with vertices, $V_i = (x_i, y_i), i = 1, 2, 3$.

Step 1: Evaluate $z_i = f(x_i, y_i)$ for $i = 1, 2, 3$ and reorder so $z_1 \le z_2 \le z_3$, where

$$\mathbf{B} = (x_1, y_1), \quad \mathbf{G} = (x_2, y_2), \quad \mathbf{W} = (x_3, y_3).$$

B is the ***best vertex***, **G** is the ***good vertex***, and **W** is the ***worst vertex***.

[2] John H. Mathews, Kurtis D. Fink (2004), *Numerical Methods Using MatLab*, 4^{th} Edition, Prentice Hall (ISBN: 0-13-065248-2)

Nelder-Mead Method

Step 2: Find the midpoint (centroid in higher dimensions) of the *Good Side*:

$$\mathbf{M} = \frac{\mathbf{B}+\mathbf{G}}{2} = \left(\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2}\right).$$

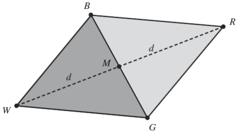
Step 3: **Reflection**: Create the line segment $\overline{\mathbf{MW}}$ through the midpoint with length d, then extend the line segment a distance d past $\mathbf{M}$ to find $\mathbf{R}$, the *reflection point*:

$$\mathbf{R} = \mathbf{M} + (\mathbf{M} - \mathbf{W}) = 2\mathbf{M} - \mathbf{W}.$$

If the *reflected point* is better than the **good vertex**, $\mathbf{G}$, but not better than the **good vertex**, $\mathbf{B}$, so

$$f(\mathbf{B}) \le f(\mathbf{R}) < f(\mathbf{G}),$$

then create the new simplex **BRG** by replacing **W** with **R** and return to **Step 1**.

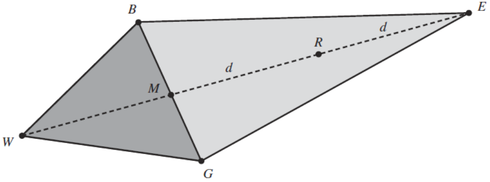


Nelder-Mead Method

Step 4: **Expansion**: If the function is smaller at $\mathbf{R}$ than $f(\mathbf{B})$, then this may be the correct direction toward the minimum, so extend the line segment another distance d to a point $\mathbf{E}$:

$$\mathbf{E} = \mathbf{R} + (\mathbf{R} - \mathbf{M}) = 2\mathbf{R} - \mathbf{M}.$$

If $f(\mathbf{E}) < f(\mathbf{R})$, then this is a better vertex than $\mathbf{R}$, so create the new simplex **BEG** by replacing **W** with **E** and return to **Step 1**.

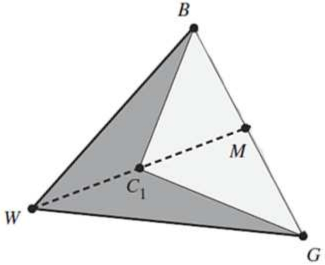


Nelder-Mead Method

Step 5: **Contraction**: If this stage is reached, then $f(\mathbf{R}) \geq f(\mathbf{G})$.

Find the midpoint of the line segment $\overline{\mathbf{MW}}$ and label this ***contracted point*** $\mathbf{C}$.

If the contracted point is better than the worst point, $f(\mathbf{C}) < f(\mathbf{W})$, then this is a better vertex than $\mathbf{W}$, so create the new simplex $\mathbf{BCG}$ by replacing $\mathbf{W}$ with $\mathbf{C}$ and return to **Step 1**.

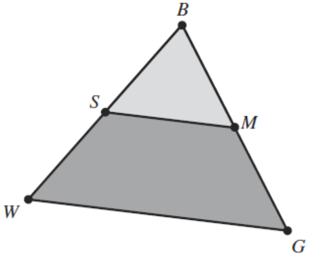


Nelder-Mead Method

Step 6: **Shrink**: If this stage is reached (which rarely occurs), then none of the created points did better than **W**.

At this stage, the **best vertex**, **B** is kept and the simplex is shrunk.

In addition to finding **M**, we find the midpoint, **S** of the line segment $\overline{\mathbf{BW}}$, then create the new simplex **BMS** by replacing **W** with **S** and **G** with **M** and return to **Step 1**.



Nelder-Mead Method

The ***initial simplex*** is important.

- If the initial simplex is too small, the solution can become trapped locally.
- The initial simplex often depends on the nature of the problem.

The ***termination*** conditions vary.

- The original paper of Nelder and Mead used the variation in the function values of the simplex.
- Alternately, one could compare the distance between the simplex nodes.
- The solution is taken as the last ***best vertex*** in the iteration.

Nelder Mead Example

Consider the function:

$$f(x_1, x_2) = (x_1 - 3)^2 + (x_2 - 4)^2$$

```
function z = fcn( x )
%function here
z=((x(1)-3)^2)+((x(2)-4)^2);
end
```

This clearly has a ***minimum*** at $(x_1, x_2) = (3, 4)$.

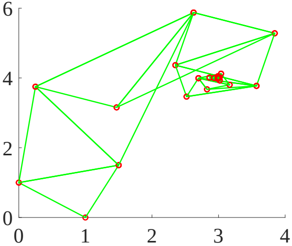
The **Nelder-Mead Algorithm** is applied to this problem with initial simplexes:

1. $V: (0,0), (1,0), (0,1)$
2. $V: (0,0), (6,0), (0,6)$

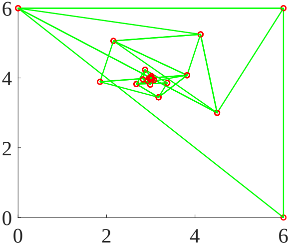
Iterations are seen on the next slide.

Nelder Mead Example

The **Nelder-Mead Algorithm** is visualized below:



$V_0 : (0,0), (1,0), (0,1)$



$V_6 : (0,0), (6,0), (0,6)$

Nelder Mead Population Model

The **population model sum of square errors** satisfies

$$E(P_0, r) = \sum_{i=0}^{n} \left(P_0 e^{rt_i} - P_i\right)^2,$$

which in **MatLab** is given by

```
function LS = malsqsim(p)
%Sum of square errors: Malthusian growth model
t = [0:10:220];
y1 = [3.929 5.308 7.24 9.638 12.866 17.069 23.192 ...
    31.443 39.818];
y2 = [50.189 62.948 76.212 92.228 106.022 122.775 ...
    132.165 150.697];
y = [y1,y2,179.323 203.302 226.546 248.71 281.422 ...
    308.746];
LS = sum((p(1)*exp(p(2)*t)-y).^2);
end
```

Nelder Mead Population Model

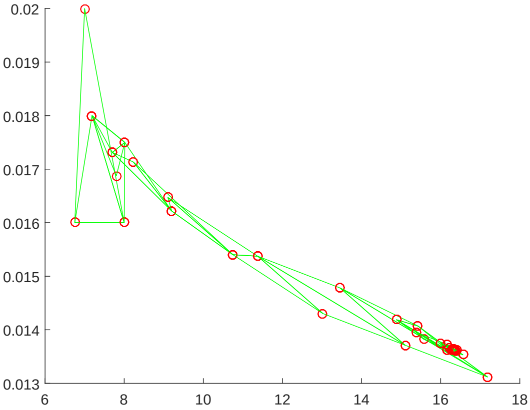
The **Nelder-Mead Algorithm** is applied to this problem with initial simplex starting near the point from the ***linear least squares best fit*** with

$$(P_0, r) = (6.32, 0.0196).$$

Iterations are seen on the next slide.

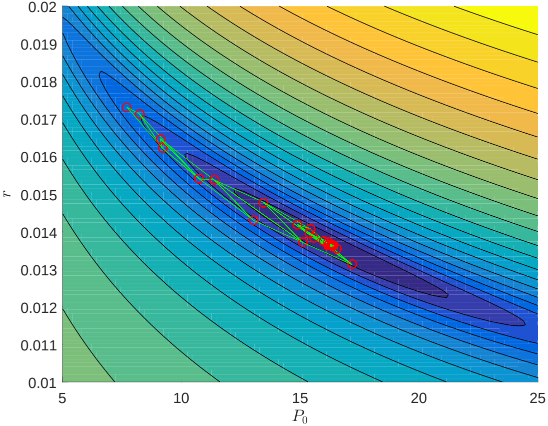
Nelder Mead Population Model

The **Nelder-Mead Algorithm** is visualized below:



Nelder Mead Population Model

When **Nelder-Mead iteration** is overlayed on the original contour graph, we see the following:



Fitting the Population Model with MatLab

Recall that the function we want to minimize is the **nonlinear sum of square errors**:

$$E(P_0, r) = \sum_{i=0}^{n} \left(P_0 e^{rt_i} - P_i\right)^2$$

In MatLab

```
function LS = mallstsq(p,t,y)
%Least Squares sum of square errors to Malthusian ...
    growth model
LS = sum((p(1)*exp(p(2)*t)-y).^2);
end
```

We apply the MatLab function `fminsearch`, using the multidimensional unconstrained nonlinear minimization (Nelder-Mead).

Fitting the Population Model with MatLab

The output from the MatLab command

```
[p,err]=fminsearch(@mallstsq,p0,[],t,pop)
```

is $p = [16.345612, 0.0136284]$ and $err = 2875.53$

This gives the **Nonlinear Least Squares Best Model**

$$P(t) = 16.345612e^{0.0136284t},$$

with a substantially improved sum of square error $= 2875.53$