Newton Solver for Constrained 2-dimensional Problems: With an application to cubic splines in dynamic programming*

Simon Mongey†
New York University

November 21, 2016

1 Introduction

In this note I present a simple method for using a Newton algorithm for solving a two dimensional maximization problem of the type

$$\max_{x,y} F(x,y) \quad \text{subject to } x \in [\underline{x}, \bar{x}]\,,\; y \in \left[\underline{y}, \bar{y}\right].$$

where $F(x,y)$ is a twice-continuously differentiable function. This is a common problem in dynamic programming where $F(x,y)$ is - conditional on current states - the sum of a flow pay-off function $f(x,y)$ and a discounted expected value such as in the following general

*This note was written in the process of writing Mongey Williams (2016). The formulation of the one dimension, one constraint problem owes to Victor Tsyrrenikov in a second year computational economics course taught at NYU in 2014. The accompanying code represents trimmed down versions of code in Mario Miranda and Paul Fackler's *Applied Computational Economics and Finance* which is distributed in the *Compecon* website: http://www4.ncsu.edu/∼pfackler/compecon/toolbox.html. The only reference within is to Garcia, C. and W. Zangwill (1981): *Pathways to Solutions, Fixed Points, and Equilibria*, Prentice Hall.

†simon.mongey@nyu.edu19 W 4th St, New York City, NY 10012.

problem

$$\begin{aligned} V(x,y,z) &= \max_{x',y'} \underbrace{f(x,y,z,x',y') + \beta V^e(x',y',z)}_{F(x',y')} \\ \text{s.t.} & \\ (x',y') &\in [\underline{x}(x,y,z), \bar{x}(x,y,z)] \times \left[\underline{y}(x,y,z), \bar{y}(x,y,z)\right] \\ V^e(x,y,z) &= \int_\varepsilon V(g_x(s,\varepsilon), g_y(s,\varepsilon), g_z(s,\varepsilon))\, dH(\varepsilon|x,y,z). \end{aligned}$$

Despite being a common optimization problem, it seems like there are few robust tools for solving the problem using a Newton scheme. The main problems posed are (i) the constraints on the choice variables, (ii) the properties of the derivatives of the continuation value $V^e(x,y)$. Most descriptions of Newton solvers consider the unconstrained case, or a practitioner will be using a linear approximation scheme for the expected value. This leads to the use of golden search or Nelder-Mead methods that, while highly efficient in one dimension, are incredibly slow in two dimensions. Here I show that if one has a cubic approximation for $V^e(x,y)$ and uses a modification of the technique proposed by Garcia-Zangwill to extend their approach to two dimensions then a very fast Newton scheme approach can be used. Importantly, when solving a recursive problem the solutions to the problem above may be very similar over successive iterations and as opposed to Nelder-Mead and Golden search, the Newton method can use the previous solution as a guess, leading to accelerating solutions as the value function approaches convergence.

I first consider a one-dimensional problem with a one-sided constraint, followed by a one-dimensional problem with a two-sided constraint and then the two dimensional problem with two constraints. I then consider the case where an approximation is available to a continuation value that has been constructed with cubic B-splines as per de Boor's algorithm.

2 One variable, one constraint

Let's consider a problem of the form

$$\max_x \; F(x) \quad \text{subject to } x \in \left[\underline{x}, \infty\right)$$

where F has continuous first and second derivatives. The Lagrangean for this problem is

$$\mathcal{L}(x) = F(x) + \lambda_x \left(x - \underline{x}\right)$$

with Karush-Kuhn-Tucker (KKT) conditions

$$\begin{aligned} F'(x) + \lambda_x &= 0 \\ x &\geq \underline{x} \\ \lambda_x &\geq 0 \end{aligned}$$

which can be written

$$\begin{aligned} F'(x) + \lambda_x &= 0 \\ \lambda_x(x - \underline{x}) &= 0. \end{aligned}$$

One could solve this system in λ and x but finds the issue that $\lambda_x \geq 0$ and $x \geq \underline{x}$. The idea here is to re-write the problem as a function of *one* variable m which lies on the real line. Crucially we require that both λ and x are once continuously differentiable in m so that we can apply the Newton algorithm to m. The following two functions satisfy this requirement and the conditions above

$$\begin{aligned} \lambda_x(m) &= (\max\{m, 0\})^2 \\ x &= \underline{x} + (\max\{-m, 0\})^2 \end{aligned}$$

If $m > 0$, then $\lambda_x(m) = m^2$ and $x = \underline{x}$, while if $m < 0$ then $\lambda_x = 0$ and $x = \underline{x} + m^2$. If we check the above conditions we see that all values of m imply values that satisfy the joint restrictions on (λ_x, x). Satisfying these constraints we then aim to solve the zero-system given by the first-order condition of the Lagrangian in terms of m

$$L(m) = F'(x(m)) + \lambda_x(m) = 0$$

We can apply the Newton solver to this system since it is smooth in m. Given an initial guess m_0 we have the usual update rule

$$m_{k+1} = m_k - L'(m_k)^{-1} L(m_k).$$

Note that this is also simple to compute. If $m < 0$ we have $x'(m)$ so $L'(m) = 2m$ and $L(m) = F'(\underline{x}) + m^2$, while if $m \geq 0$ we have $L'(m)$ then by the chain-rule we have $L'(m) = F''(x(m))x'(m) = 2m \times F'(x(m))$ and $L(m) = F'(x(m))$.

In the case where F has no closed form then we use centered numerical derivatives

around $x(m)$ for small h

$$\begin{aligned} F'(x(m)) &\approx \frac{F(x(m)+h) - F(x(m)-h)}{2h} \\ F''(x(m)) &\approx \frac{F(x(m)+h) - 2F(x(m)) - F(x(m)-h)}{4h^2}. \end{aligned}$$

Note that at $x = \underline{x}$ it may be the case that the centered derivative can't be computed since $F(\underline{x} - h)$ is not defined (for example if $\underline{x} = 0$ and the function F includes a log function). In this case we would use the right-hand derivative at the boundary

$$F'(\underline{x}) \approx \frac{F(\underline{x}+h) - F(\underline{x})}{2h}.$$

At the $x = \underline{x}$ note that we don't have to compute the second derivative since $x'(m) = 0$.

In this case and what follows we are trying to minimize on the number of evaluations of F per iteration of the solver, in this case we have to compute F a maximum of three times. Note that we may also save some work by computing $F'(\underline{x})$ in advance.

3 One variable - two constraints

The idea here is the same as the above, except with a different functional form mapping m to the choice variable and multipliers that again satisfies the first order conditions and continuity in the first derivative. I propose one particularly easy function to compute including the derivatives with respect to m, however numerous others may satisfy these requirements.

The problem is

$$\max_x \; F(x) \quad \text{subject to } x \in \left[\underline{x}, \bar{x}\right)$$

where F has continuous first and second derivatives. The Lagrangian for this problem is

$$\mathcal{L}(x) = F(x) + \lambda_x^1 (x - \underline{x}) + \lambda_x^2 (\bar{x} - x)$$

with Karush-Kuhn-Tucker (KKT) conditions

$$\begin{aligned} F'(x) + \lambda_x^1 - \lambda_x^2 &= 0 \\ x &\geq \underline{x}, \quad x \leq \bar{x} \\ \lambda_x^1, \lambda_2^x &\geq 0 \end{aligned}$$

which can be written

$$\begin{aligned} F'(x) + \lambda_x^1 - \lambda_x^2 &= 0 \\ \lambda_x^1 (x - \underline{x}) &= 0 \\ \lambda_x^2 (\bar{x} - x) &= 0. \end{aligned}$$

We again specify a set of functions $x(m)$, $\lambda_x^1(m)$, $\lambda_x^2(m)$ that have a continuous first derivative in m and satisfy these conditions. The function we specify is

$$\left(x, \lambda_x^1, \lambda_x^2\right) = \begin{cases} \left(\underline{x},\ m^2,\ 0\right) & , \text{if } m < 0 \\ \left(\underline{x} + am^2,\ 0,\ 0\right) & , \text{if } m \in \left[0, \frac{1}{2}\right) \\ \left(\bar{x} - a\left(1-m\right)^2,\ 0,\ 0\right) & , \text{if } m \in \left[\frac{1}{2}, 1\right] \\ \left(\bar{x},\ 0,\ \left(m-1\right)^2\right) & , \text{if } m > 1 \end{cases}, \quad a = 2 \times (\bar{x} - \underline{x}) \tag{1}$$

Essentially this splices together a convex quadratic function with it's concave reflection to get x from $\underline{x}$ at $m = 0$ to $\bar{x}$ at $m = 1$ with derivatives $x'(0) = 0$, $x'(1) = 0$ and a continuous right and left derivatives at $m = \frac{1}{2}$. The properties of the Lagrange multipliers is looked after in the same way as the one-sided constraint.

As above we then use the Newton algorithm applied to the zero system $L(m) = 0$ in m where

$$\begin{aligned} L(m) &= F'(x(m)) + \lambda_x^1(m) - \lambda_x^2(m). \\ \nabla L(m) &= F''\left(x(m)\right) \frac{\partial x(m)}{\partial m} + \frac{\partial \lambda_x^1(m)}{\partial m} - \frac{\partial \lambda_x^2(m)}{\partial m} \\ \nabla L(m) &= \begin{cases} \frac{\partial \lambda_x^1(m)}{\partial m} = 2m & , \quad m \leq 0 \\ F''\left(x(m)\right) \frac{\partial x(m)}{\partial m} & , \quad m \in (0,1) \\ -\frac{\partial \lambda_x^2(m)}{\partial m} = -2(m-1) & , \quad m < 0 \end{cases} \end{aligned}$$

and

$$\frac{\partial x(m)}{\partial m} = \begin{cases} 2am & , \quad m \in (0, 0.5) \\ 2a(1-m) & , \quad m \in [0.5, 1) \end{cases}$$

The numerical derivatives of F are computed numerically according to whether one is at the boundary. Note that the derivatives of x and λ with respect to m are exact, these are the

only ones approximated.

$$F'(x(m)) = \begin{cases} \left(\frac{1}{h}\right)\left[F(x(m)+h) - F(\underline{x})\right] & , \quad m \leq 0 \\ \left(\frac{1}{2h}\right)\left[F(x(m)+h) - F(x(m)-h)\right] & , \quad m \in (0,1) \\ \left(\frac{1}{h}\right)\left[F(x(m)) - F(x(m)-h)\right] & , \quad m < 0 \end{cases}$$

and the second derivative which only needs to be computed for the interior cases since $x'(m) = 0$ for the boundary cases

$$F''(x(m)) = \left(\frac{1}{h^2}\right)\left[F(x(m)+h) - 2F(x(m)) + F(x(m)-h)\right]$$

3.1 More complicated constraints

Suppose that the problem is

$$\max_x \; F(x) \quad \text{subject to } x \leq \bar{x} \text{ and } \frac{1}{x} \geq a$$

which is a specific case of the problem where a constraint is of the form $g(x) \geq 0$ where $g(x)$ is convex so that the problem remains convex. In this case the Lagrangean is

$$\mathcal{L}(x) = F(x) + \lambda_x^1 \left(\frac{1}{x} - a\right) + \lambda_x^2 \left(\bar{x} - x\right)$$

with KKT conditions

$$\begin{aligned} F'(x) - \lambda_x^1 \left(\frac{1}{x^2}\right) - \lambda_x^2 &= 0 \\ \frac{1}{x} &\geq a, \quad x \leq \bar{x} \\ \lambda_x^1, \lambda_2^x &\geq 0 \end{aligned}$$

Here we can use the same strategy as above, except when $m > 1$ we set $x(m) = \frac{1}{a}$ in (1) and the condition we solve for m is

$$F'(x(m)) - \lambda_x^1(m) \left(\frac{1}{x(m)^2}\right) - \lambda_x^2(m) = 0.$$

4 Two variables - two constraints

Here we proceed as above using the Newton method to solve a two equation system in two unknowns m_x and m_y. The problem is

$$\max_{x,y} F(x,y) \quad \text{subject to } x \in [\underline{x}, \bar{x}],\ y \in \left[\underline{y}, \bar{y}\right]$$

The Lagrangian for the problem is

$$\mathcal{L}(x,y) = F(x,y) + \lambda_x^1 (x - \underline{x}) + \lambda_x^2 (\bar{x} - x) + \lambda_y^1 \left(y - \underline{y}\right) + \lambda_y^2 (\bar{y} - y)$$

with the first order conditions

$$\begin{aligned} F_x(x,y) + \lambda_x^1 - \lambda_x^2 &= 0 \\ F_x(x,y) + \lambda_y^1 - \lambda_y^2 &= 0 \end{aligned}$$

and the standard set of complementary slackness conditions. We specify a set of six functions $x(m_x)$, $\lambda_x^1(m_x)$, $\lambda_x^2(m_x)$, $y(m_y)$, $\lambda_y^1(m_y)$, $\lambda_y^2(m_y)$ in the same way as for the one-variable two-sided constraint problem. This gives us the zero system $L(m_x, m_y) = 0$ that we solve using the Newton method

$$\begin{aligned} F_x(x(m_x), y(m_y)) + \lambda_x^1(m_x) - \lambda_x^2(m_x) &= 0, \\ F_y(x(m_x), y(m_y)) + \lambda_y^1(m_y) - \lambda_y^2(m_y) &= 0. \end{aligned}$$

Given a guess $\mathbf{m}^0 = \left(m_x^0, m_y^0\right)'$ the iterative scheme is

$$\begin{bmatrix} m_x^{k+1} \\ m_y^{k+1} \end{bmatrix} = \begin{bmatrix} m_x^k \\ m_y^k \end{bmatrix} - \nabla L\left(m_x^k, m_y^k\right)^{-1} \times L\left(m_x^k, m_y^k\right)$$

where

$$\begin{aligned} \nabla L(m_x, m_y)^{-1} &= \begin{bmatrix} F_{xx}(x(m_x), y(m_y))x'(m_x) + \lambda_x^{1\prime}(m_x) - \lambda_x^{2\prime}(m_x) & F_{xy}(x(m_x), y(m_y))y'(m_y) \\ F_{xy}(x(m_x), y(m_y))x'(m_x) & F_{yy}(x(m_x), y(m_y))y'(m_y) + \lambda_y^{1\prime}(m_y) - \end{bmatrix} \\ L\left(m_x, m_y\right) &= \begin{bmatrix} F_x(x(m_x), y(m_y)) + \lambda_x^1(m_x) - \lambda_x^2(m_x) \\ F_y(x(m_x), y(m_y)) + \lambda_y^1(m_y) - \lambda_y^2(m_y) \end{bmatrix}. \end{aligned}$$

Note that the off-diagonal elements of the Jacobian *are not* identical since the Jacobian is computed with respect to m_x and m_y, not x and y. If $m_x \notin (0,1)$ then $x'(m_x) = 0$ and the

bottom-left entry of the Jacobian zero. Similarly regarding the top-left entry and m_y. This can save on computations.

In the case in which one is computing the derivatives and second derivatives of F numerically the most expensive derivative to compute is the cross-partial derivative. In this case it is recommended to use the following approximation

$$F_{xy}(x,y) \approx \frac{1}{4h^2}\left(F(x+h,y+h) - F(x+h,y) - F(x,y-h) + 2F(x,y) - F(x-h,y) - F(x,y-h) + F(\right.$$

Since all but $F(x+h,y+h)$ and $F(x-h,y-h)$ are required to compute the first partial derivatives F_x and F_y to evaluate $L(m_x,m_y)$, then this requires only an additional two evaluations of F rather than the amount four additional evaluations required under the more standard formula

$$F_{xy}(x,y) \approx \frac{1}{4h^2}\left(F(x+h,y+h) - F(x+h,y-h) - F(x-h,y+h) + F(x-h,y-h)\right).$$

Again note that the derivatives of $x(m_x)$ and $y(m_y)$ as well as the derivatives of the multipliers with respect to m's are available in closed form.

An issue arises when, for example $m_x < 0$ and $m_y \in (0,1)$ in this case the bottom-left entry is zero since $x'(m_x) = 0$ but we still need to compute $F_{xy}(x(m_x), y(m_y))$ for the top-right entry. With $x = \underline{x}$ it could be the case that$F(x-h,y)$ is not defined. We therefore first compute $F_x(x,y)$ numerically using a *right* derivative which does exist, and then differentiate that numerical derivative to ge $F_{xy}(x,y)$. There are four such cases that we need to consider and I provide the exact expressions for these derivative in Appendix A.

5 Dynamic programming with cubic splines

The basic problem we're considering is the optimization step in each iteration of the solution of a dynamic programming problem of the form

$$\begin{aligned} V(s) &= \max_{(x',y')\in[\underline{x},\bar{x}]\times\left[\underline{y},\bar{y}\right]} f(s,x',y') + V^e(x',y',z) \\ V^e(s) &= \beta \int V(g_x(s,\varepsilon), g_y(s,\varepsilon), g_z(s,\varepsilon) dF(\varepsilon|s). \\ s &= (x,y,z) \end{aligned}$$

In this problem we seek to solve for the value function V and the expected value function V^e given the flow pay-off function f, the transition functions g_x, g_y and g_z and the distribution of the vector of stochastic variables ε where the distribution for ε depends on the state-variables

x, y, z. A standard example may be a firm dynamic investment problem under stochastic productivity in which case we may have a firm with state variables of two types of capital k_a and k_b and productivity z with

$$\begin{aligned}
f(s, k'_a, k'_b) &= e^z k_a^\alpha k_b^{1-\alpha} - \phi_a \left(\frac{k'_a - (1-\delta_a)k_a}{k_a} \right)^2 - \phi_b \left(\frac{k'_b - (1-\delta_b)k_b}{k_b} \right)^2 - [k'_a - (1-\delta_a)k_a] - [k'_b - (1 - \\
\varepsilon|s &\sim N(0, \sigma_\varepsilon) \\
g_{k_a}(s, \varepsilon) &= k_a \\
g_{k_b}(s, \varepsilon) &= k_b \\
g_z(s, \varepsilon) &= \rho z + \varepsilon
\end{aligned}$$

When solving this problem using collocation methods we would set a grid of approximation nodes $\mathbf{s}$ and then approximate $V^e(s)$ using a particular form of approximating function. I cover these in a separate note. When using a Newton-solver for the coefficients of the approximating polynomial the *iteration to convergence* of the value function problem can be very fast. The time consuming part of the problem is the optimization of (x', y') given an approximant for $V^e(s)$.

If we have $V^e(s)$ approximated using a set of basis functions that have a continuous second derivative then we can use the Newton-solver described in the preceding sections in the presence of the the constraints $(x', y') \in [\underline{x}, \bar{x}] \times \left[\underline{y}, \bar{y}\right]$. Such approximation schemes include *Chebyshev polynomials*, standard *polynomials*, and *cubic splines*. In the first two cases the derivatives of the approximant can be costly to compute and as we've seen we need to quickly be able to evaluate first and second derivatives. In the case of a *cubic spline* implemented as a *B-spline* using de Boor's algorithm these derivatives are immediately at hand. A set of other features also fall into place.

5.1 The cubic B-spline - One dimension

The cubic B-spline in one dimension is constructed as follows. Given a set of approximating nodes $\mathbf{x} = [x_0, x_1, x_2, \ldots, x_N]$ we compute the basis functions at x recursively up to $k = 3$ as follows

$$\begin{aligned}
B_{i,1}(x): &= \begin{cases} 1 & \text{if} \quad x_i \le x \le x_{i+1} \\ 0 & \text{otherwise} \end{cases} \\
B_{i,k}(x): &= \frac{x - x_i}{x_{i+k-1} - x_i} B_{i,k-1}(x) + \frac{x_{i+k} - x}{x_{i+k} - x_{i+1}} B_{i+1,k-1}(x)
\end{aligned}$$

Then given a vector of coefficients c for the existing spline we have the value of the approximant given by

$$\tilde{F}(x) = \sum_{i=1}^{N} c_i B_{i,3}(x) = B(x)c, \quad \text{where } B(x) = [B_{1,3}(x), \ldots, B_{N,3}(x)]\,,\ c = [c_1, \ldots, c_N]'$$

The incredibly neat part of this procedure is that the *first* and *second* derivatives are essentially computed on the way to computing $B(x)$. We can see this by differentiating $B_{i,3}(x)$, what we pick up is some weights plus the derivatives of $B_{i,2}(x)$, and when we differentiate $B_{i,2}(x)$ we get some weights and the derivatives of $B_{i,1}(x)$ which are zeros. The particulars of this are left to my attached code which is taken from Miranda and Fackler's *Compecon* tools, but for now we note that given the grid $\mathbf{x}$ we can compute two matrices D_1 and D_2 which are *derivative operators.* We can then compute the derivatives exactly as

$$\begin{aligned} F(x) &= B_0(x) \\ F'(x) &= D_1 B_1(x) \\ F''(x) &= D_2 B_2(x) \end{aligned}$$

where the basis matrices $\{B_0, B_1, B_2\}$ are all computed in any evaluation of F, that is we get the first and second derivatives for free each time we evaluate the approximant. This is obviously ideal for the application of the Newton-method since we need not compute costly numerical derivatives of V^e (which require numerous evaluations of the approximant) since the analytical derivatives of the approximant are at hand under one evaluation.

5.2 The cubic B-spline - Two dimensions

For a one dimensional problem we have seen that the approximant is evaluated as the matrix multiplication of a basis matrix $B_0(x)$ and a coefficient vector c. In the case of a two-dimensional approximant using B-splines this remains the case where we evaluate the approximant as $F(x, y) = B_0(x, y)c$. The neat part about approximating in two dimensions using B-splines rather than, say Chebyshev polynomials or 'saturated' polynomials, is that the spline essentially *separates* the parts of F due to x and y, that is we compute $B_0(x, y)$ as a *tensor product* of two basis matrices:

$$B_0(x, y) = B_0^y(y) \circledast B_0^x(x)$$

where $B_0^y(y)$ and $B_0^x(x)$ are evaluated in the same way as the one-dimensional spline. This implies that for a derivative such as $F_x(x,y)$ we have

$$\begin{aligned} F_x(x,y) &= \frac{\partial}{\partial x} B_0(x,y)c = \frac{\partial}{\partial x}\left[B_0^y(y) \circledast B_0^x(x)\right] c = \left[B_0^y(y) \circledast B_1^x(x)\right] c. \\ F_{xx}(x,y) &= \frac{\partial^2}{\partial x^2} B_0(x,y)c = \frac{\partial^2}{\partial x^2}\left[B_0^y(y) \circledast B_0^x(x)\right] c = \left[B_0^y(y) \circledast B_2^x(x)\right] c. \end{aligned}$$

Now when evaluating $F(x,y)$ we compute the six basis matrices $\{B_0^x, B_1^x, B_2^x, B_0^y, B_1^y, B_2^y\}$, and so after one evaluation the matrices required (with the use of tensor products) to compute all partial, cross-partial and second derivatives are immediately available to us. Again this means we never have to numerically compute the first and second order derivatives of V^erequired in the Newton algorithm for optimization over (x', y') given the state vector s.

A Computation of cross-partials

1. Consider the case where $m_x < 0$, $m_y \in (0,1)$

   - Then $x'(m_x) = 0$, but $y'(m_y) \neq 0$, so we still have to compute $F_{xy}(x,y)$.
   - However we can't compute the centered partial derivative since F at $x - h$ may not be defined.
   - We therefore proceed as follows.
   - First take the numerical *right* derivative of $F(x,y)$ with respect to x

   $$F_x(x,y) = \frac{1}{h}\left[F(x+h,y) - F(x,y)\right]$$

   - Then take this expression and apply a numerical *centered* derivative for y

   $$F_{xy}(x,y) = \frac{1}{2h^2}\left[F(x+h,y+h) - F(x,y+h) - F(x+h,y-h) + F(x,y-h)\right]$$

2. Consider the case where $m_x > 1$, $m_y \in (0,1)$

   - Then $x'(m_x) = 0$, but $y'(m_y) \neq 0$, so we still have to compute $F_{xy}(x,y)$.
   - However we can't compute the centered partial derivative since F at $x + h$ may not be defined.
   - We therefore proceed as follows.

- First take the numerical *left* derivative of $F(x, y)$ with respect to x

$$F_x(x, y) = \frac{1}{h} [F(x, y) - F(x - h, y)]$$

- Then take this expression and apply a numerical *centered* derivative for y

$$F_{xy}(x, y) = \frac{1}{2h^2} [F(x, y + h) - F(x - h, y + h) - F(x, y - h) + F(x - h, y - h)]$$

3. Consider the case where $m_y < 0$, $m_x \in (0, 1)$

- Then $x'(m_x) \neq 0$, but $y'(m_y) = 0$, so we still have to compute $F_{xy}(x, y)$.
- However we can't compute the centered partial derivative since F at $y - h$ may not be defined.
- We therefore proceed as follows.
- First take the numerical *right* derivative of $F(x, y)$ with respect to y

$$F_y(x, y) = \frac{1}{h} [F(x, y + h) - F(x, y)]$$

- Then take this expression and apply a numerical *centered* derivative for x

$$F_{xy}(x, y) = \frac{1}{2h^2} [F(x + h, y + h) - F(x + h, y) - F(x - h, y + h) + F(x - h, y)]$$

4. Consider the case where $m_y > 1$, $m_x \in (0, 1)$

- Then $x'(m_x) \neq 0$, but $y'(m_y) = 0$, so we still have to compute $F_{xy}(x, y)$.
- However we can't compute the centered partial derivative since F at $y + h$ may not be defined.
- We therefore proceed as follows.
- First take the numerical *left* derivative of $F(x, y)$ with respect to y

$$F_y(x, y) = \frac{1}{h} [F(x, y) - F(x, y - h)]$$

- Then take this expression and apply a numerical *centered* derivative for x

$$F_{xy}(x, y) = \frac{1}{2h^2} [F(x + h, y) - F(x + h, y - h) - F(x - h, y) + F(x - h, y - h)]$$