Conjugate Gradient Methods for Multidimensional Optimization

Stephen D. Butalla & Valerie Kobzarenko

October 7, 2019

CSE-5400

Florida Institute of Technology

General Picture

- Recall the Taylor expansion of a function $f(x)$ around point $\mathbf{P}$:

$$
\begin{aligned}
f(\mathbf{x}) &\approx f(\mathbf{P}) + \sum_i \frac{\partial f}{\partial x_i} x_i + \frac{1}{2} \sum_{i,j} \frac{\partial^2 f}{\partial x_i \partial x_j} x_i x_j + \cdots \\
&\approx c - \mathbf{b} \cdot \mathbf{x} + \frac{1}{2} \mathbf{x} \cdot \mathbb{A} \cdot \mathbf{x}
\end{aligned}
$$

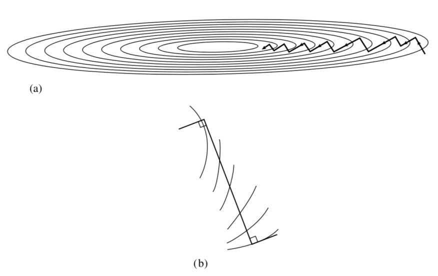
  which is (approximately) quadratic, with

$$
c \equiv f(\mathbf{P}), \qquad \mathbf{b} \equiv -\nabla f|_{\mathbf{P}}, \qquad A_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j} x_i x_j \bigg|_{\mathbf{P}}
$$

- Gradient produces N parameters; we can reduce the order of the line minimizations from direction set methods $\left(\mathcal{O}(N^2) \to \mathcal{O}(N)\right)$ if we use gradients

Naïve First Guess at an Algorithm

- At point $\mathbf{P}_0$, calculate $-\nabla f(\mathbf{P}_0)$ and begin the line search along the "steepest" changing direction
- Recalculating the gradient at each point $\mathbf{P}_i$ and continue until minimum is reached
- Known as the Steepest Descent Method
- Results in making many small steps because the gradient at the new point $\mathbf{P}_{i+1}$ will result in an orthogonal direction change



Steepest Descent Method. (a) A long, narrow valley, (b) the resulting orthogonal direction change [1].

Conjugate Gradient: Fletcher-Reeves Method

- More efficient method: Conjugate Gradient Method
- Couple the conjugate direction method like in Powell's Method, but use gradient information instead of
- Use two sets of vectors that form a recurrence relation:

$$\mathbf{g}_{i+1} = \mathbf{g}_i - \lambda_i \mathbb{A} \cdot \mathbf{h}_i, \qquad \mathbf{h}_{i+1} = \mathbf{g}_{i+1} + \gamma_i \mathbf{h}_i, \quad i = 0, 1, 2, \cdots$$

where $\mathbf{g}_i,\ \mathbf{h}_i$ satisfy the conjugate direction relations

$$\mathbf{g}_i \cdot \mathbf{g}_j = 0, \qquad \mathbf{h}_i \cdot \mathbb{A} \cdot \mathbf{h}_j = 0, \qquad \mathbf{g}_i \cdot \mathbf{h}_j = 0, \quad i < j$$

and where

$$\lambda_i = \frac{g_i^2}{\mathbf{h}_i \cdot \mathbb{A} \cdot \mathbf{h}_i} = \frac{\mathbf{g}_i \cdot \mathbf{h}_i}{\mathbf{h}_i \cdot \mathbb{A} \cdot \mathbf{h}_i}, \qquad \gamma_i = \frac{\mathbf{g}_{i+1} \cdot \mathbf{g}_{i+1}}{\mathbf{g}_i \cdot \mathbf{g}_i}$$

Polak-Ribiere Method

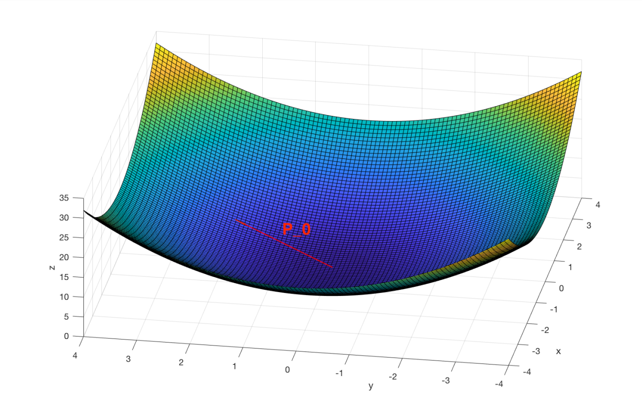
- Polak and Ribiere modified the Fletcher-Reeves algorithm
- Set
$$\gamma_i = \frac{(\mathbf{g}_{i+1} - \mathbf{g}_i) \cdot \mathbf{g}_{i+1}}{\mathbf{g}_i \cdot \mathbf{g}_i}$$
- There is some evidence that choosing γ_i above is more efficient
- Note that the function and its gradient must be known to use this method

Basic Polak-Ribiere Example

- Define your function and compute its gradient:

$$f(x, y) = x^2 + y^2$$
$$\nabla f = 2(\hat{x}x + \hat{y}y)$$

- Provide (arbitrary) initial guess of minimum: $\mathbf{P}_0^T = (1\ 2)$
- Define tolerance τ and a tolerance to distinguish between zero and a very small number, ν



$f(x, y) = x^2 + y^2$ and vector $\mathbf{P}_0$, produced in `MATLAB`

Basic Polak-Ribiere Example

- Calculate gradient at $\mathbf{P}_0^2 = (1\ 2)$

$$f(x, y) = x^2 + y^2$$
$$\nabla f|_{\mathbf{P}_0} = \hat{x}2(1) + \hat{y}2(2) = \hat{x}2 + \hat{y}4$$

- Assign components of the negative gradient to the **g** and **h** vectors

$$\mathbf{g}_i = \mathbf{h}_i = -\nabla f|_i$$

- Use the line minimization algorithm to find the minimum along the first direction, which returns $f(\mathbf{P}_{\text{lin min}})$
- If

$$2|f(\mathbf{P}_{\text{lin min}}) - f(\mathbf{P}_0)| \leq \tau\left(|f(\mathbf{P}_{\text{lin min}})| + |f(\mathbf{P}_0)| + \nu\right)$$

  minimum is at $f(\mathbf{P}_{\text{lin min}})$, return vector $\mathbf{P}_{\text{lin min}}$
- If this statement is not true, assign $\mathbf{P}_1 = \mathbf{P}_{\text{lin min}}$

Basic Polak-Ribiere Example

```
const Int ITMAX=200; //maximum iterations
const Doub EPS=1.0e-18; //used to test if function min is zero
const Doub GTOL=1.0e-8; //tolerance for zero gradient
Doub gg,dgg;
Int n=pp.size(); //number of dimensions
p=pp;//reassign vector
VecDoub g(n),h(n); //initialize g and h vectors
xi.resize(n);
Doub fp=func(p); //copy of original function at p
func.df(p,xi); //calculate gradient and return in vector xi
for (Int j=0;j<n;j++) {
 g[j] = -xi[j]; //negative gradient
 xi[j]=h[j]=g[j]; //flip gradient sign and store in h and gradient storage
}
//Main loop for minimization
for (Int its=0;its<ITMAX;its++) {
 iter=its;
 fret=linmin(); //calls linmin for line minimization
 if (2.0*abs(fret-fp) <= ftol*(abs(fret)+abs(fp)+EPS))
  return p;
fp=fret;
func.df(p,xi);
}
```

Basic Polak-Ribiere Example

- Test to see if gradient is zero (minimum)
- Define:

$$\beta_j = \frac{|\nabla \mathbf{P}_i^{(j)}| * \max(|\mathbf{P}_i^{(j)}|, 1)}{\max(|f(\mathbf{P}_i)|, 1)}$$

- If $\beta_i > 0$, compare with the tolerance variable ν defined earlier

```
Doub test=0.0; //Initialize zero gradient test value
Doub den=MAX(abs(fp),1.0);
for (Int j=0;j<n;j++) {
  Doub temp=abs(xi[j])*MAX(abs(p[j]),1.0)/den;
  if (temp > test) test=temp;
}
if (test < GTOL) return p;
```

- β_i is sufficiently small, the minimum has been found
- Loop breaks here. If not, continue to updating coefficients

Basic Polak-Ribiere Example

- Update coefficients:

  Calculate γ

  $$\gamma = \frac{\sum_i (\nabla f|_i + g_i) \nabla f|_i}{\sum_i g_i^2}$$

  Reassign values of the vector that stores the gradient and each component of $\mathbf{h}_i$:

  $$\mathbf{h}_i = \mathbf{g}_i + \gamma \mathbf{h}_i$$

- Restart line search at the new point $\mathbf{P}_1 = \mathbf{P}_{\text{lin min}}$

Basic Polak-Ribiere Example

```
//Calculate components necessary for gamma
for (Int j=0;j<n;j++) {
  gg += g[j]*g[j];
  //dgg += xi[j]*xi[j]; //Fletcher-Reeves
  dgg += (xi[j]+g[j])*xi[j];//Polak-Robiere
}
if (gg == 0.0)//if gradient is zero, return point p
  return p;
  Doub gam=dgg/gg;
  for (Int j=0;j<n;j++) {
    g[j] = -xi[j];
        xi[j]=h[j]=g[j]+gam*h[j];
}
```

Main minimization loop then restarts

Eventually, at one of the tests between the loops, the gradient will be sufficiently close to zero

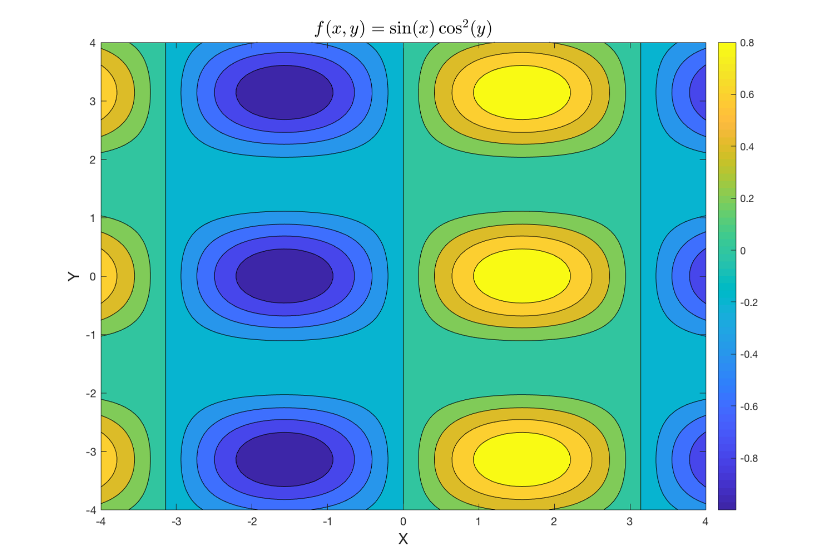
Minimum returned from this example is $(0, 0)$

$$f(0, 0) = 0$$

Code Demonstration

- Let's find the minimum of the function

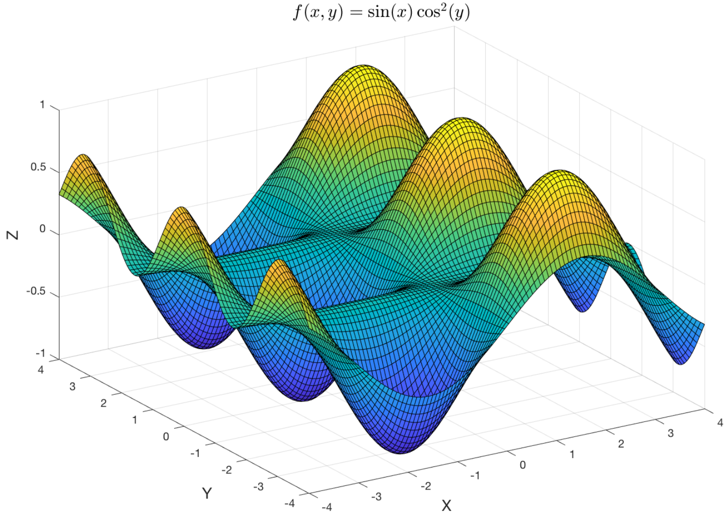
$$f(x, y) = \sin(x)\cos^2(y)$$



Code Demonstration

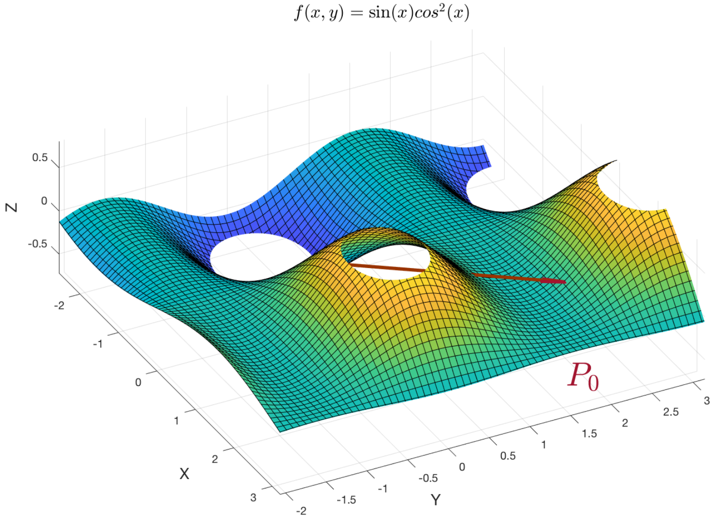
- The gradient is

$$\nabla f(x,y) = \hat{x}\cos(x)\cos^2(y) - \hat{y}2\cos(y)\sin(x)\sin(y)$$



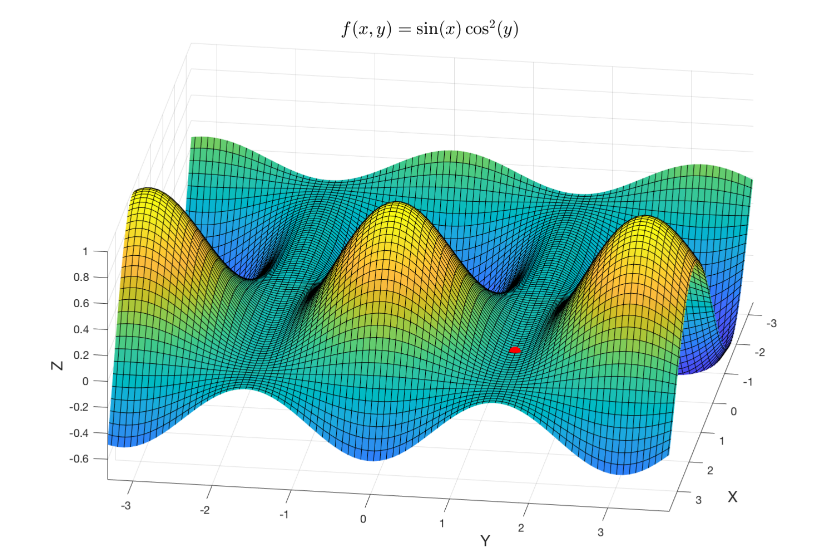
Code Demonstration

- Initial guess: $\mathbf{P}_0 = 2(\hat{x} + \hat{y})$



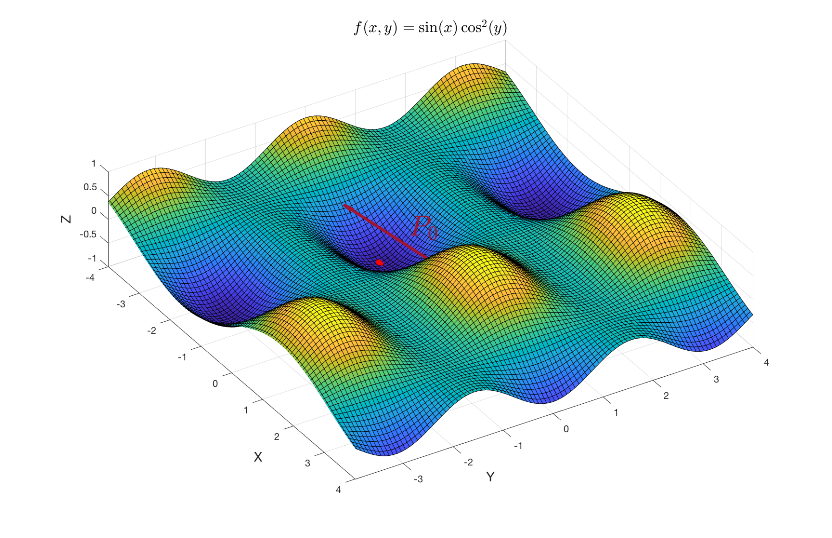
Code Demonstration

- Initial guess: $\mathbf{P}_0 = 2(\hat{x} + \hat{y})$
- Returned minimum: $(2.0, 1.57, 0)$



Code Demonstration

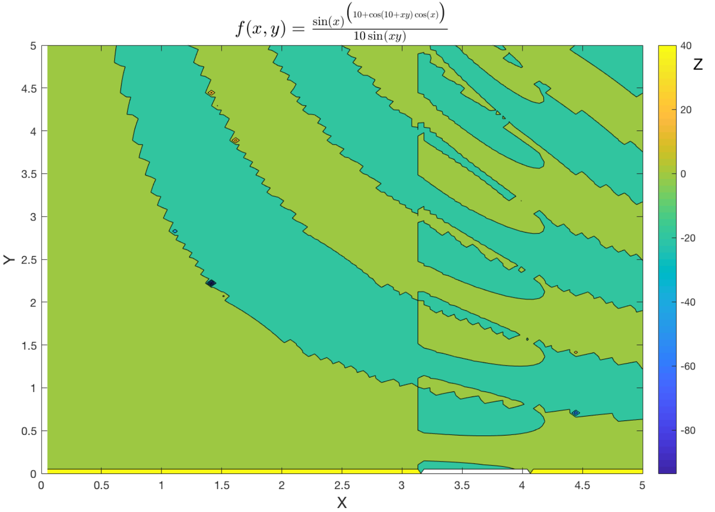
- Initial guess: $\mathbf{P}_0 = -\hat{x}2.75$
- Returned minimum: $(-1.58, 0, -1)$



Code Demonstration

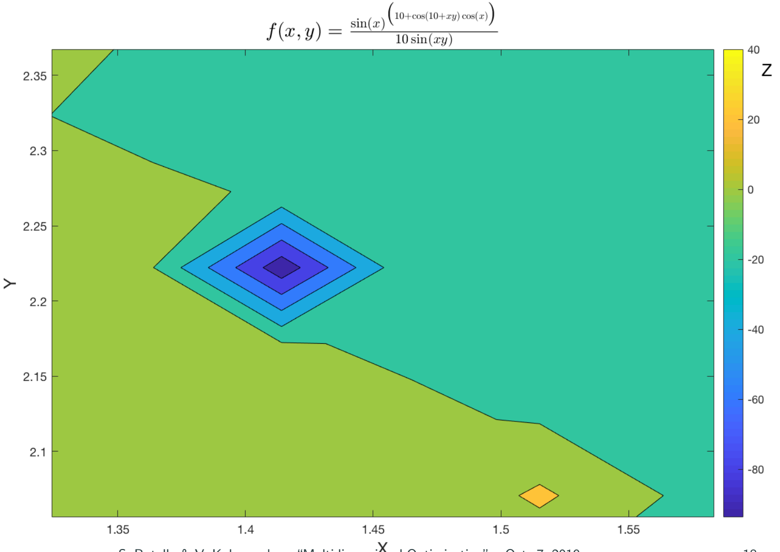
- Find the minimum of:

$$f(x,y) = \frac{\sin(x)^{\left(10+\cos(10+xy)\cos(x)\right)}}{10\sin(xy)}$$



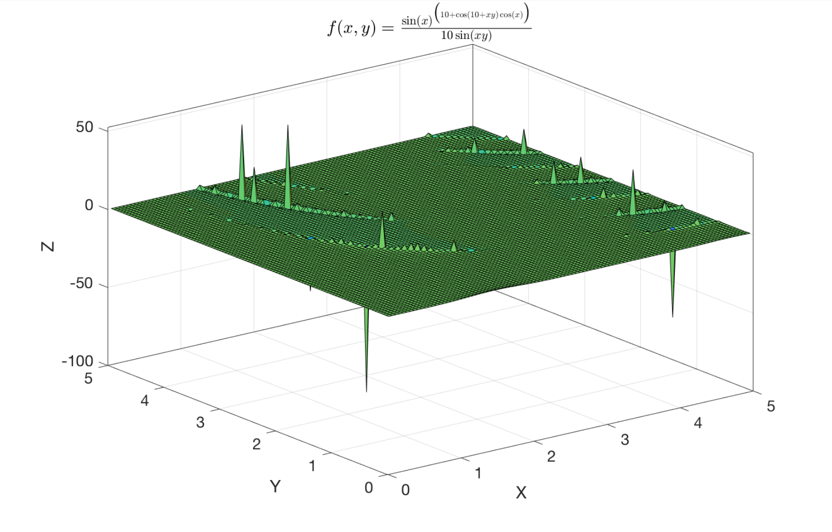
Code Demonstration

- If we zoom, a local minimum (on the domain) appears around $(x, y) = (1.5, 2.25)$



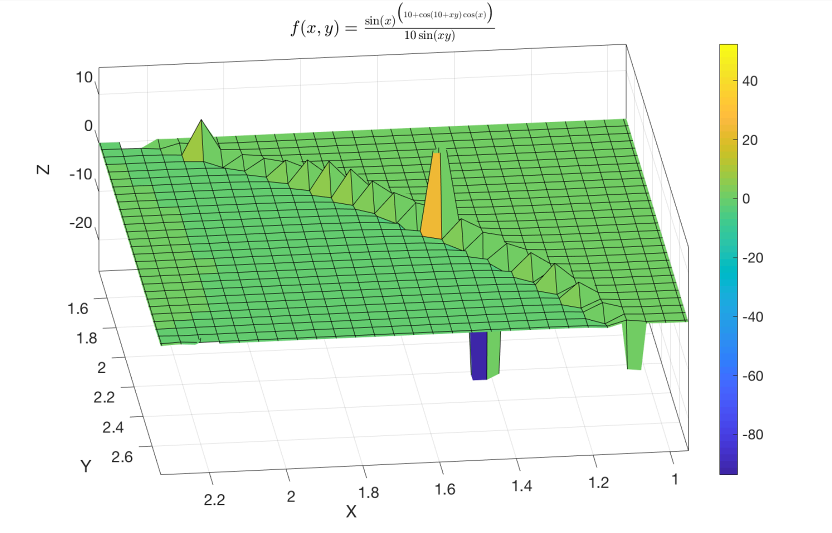
Code Demonstration

- In 3 dimensions, we see more structure:



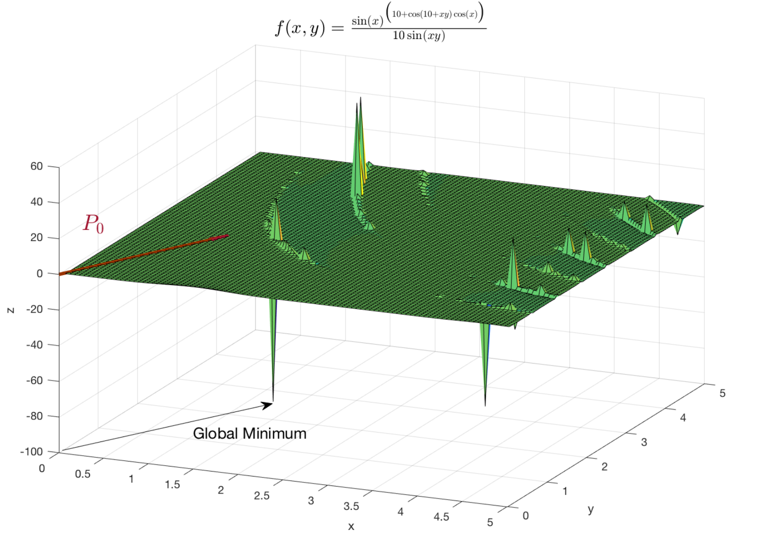
Code Demonstration

- Zoomed near the point of the minimum:



Code Demonstration

- Choose an initial guess at $\mathbf{P}_0 = \hat{x} + \hat{y}2$



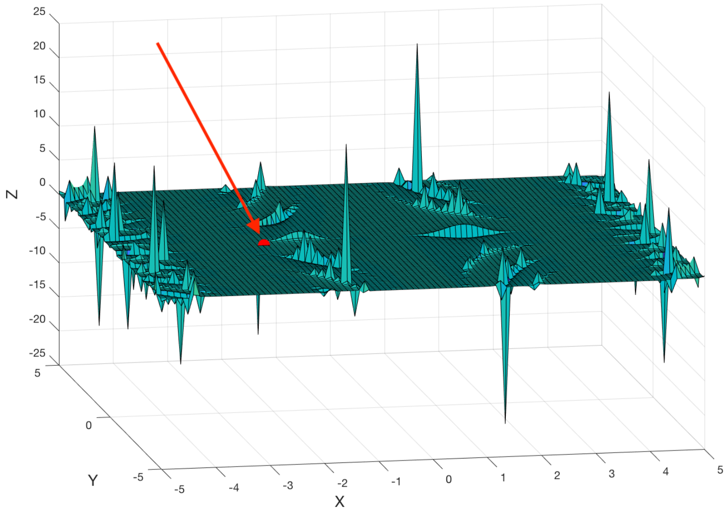
Code Demonstration

- Compute the gradient:

$$
\begin{aligned}
\nabla f(x,y) = \hat{x}\Bigg[& \frac{1}{10}\csc(xy)\sin(x)^{(\cos(x)\cos(xy+10)+10)}\Bigg((\cot(x)(cos(x)cos(xy+10)+10) \\
& + \log(\sin(x))\big(\sin(x)(-\cos(xy+10)) - y\cos(x)\sin(xy+10)\big)\Bigg) \\
& - \frac{1}{10}y\cot(xy)\csc(xy)\sin(x)^{(\cos(x)\cos(xy+10)+10)}\Bigg] \\
& + \hat{y}\Bigg[-\frac{1}{10}x\cot(xy)\csc(xy)\sin(x)^{(\cos(x)\cos(xy+10)+10)} \\
& - \frac{1}{10}x\cos(x)\sin(xy+10)\csc(xy)\log(sin(x))\sin(x)^{(\cos(x)\cos(xy+10)+10)}\Bigg]
\end{aligned}
$$

Code Demonstration

- Odd behavior results for very complex functions
- Input: (1.5, 2.5); returned minimum: (-2.24, -0.41)



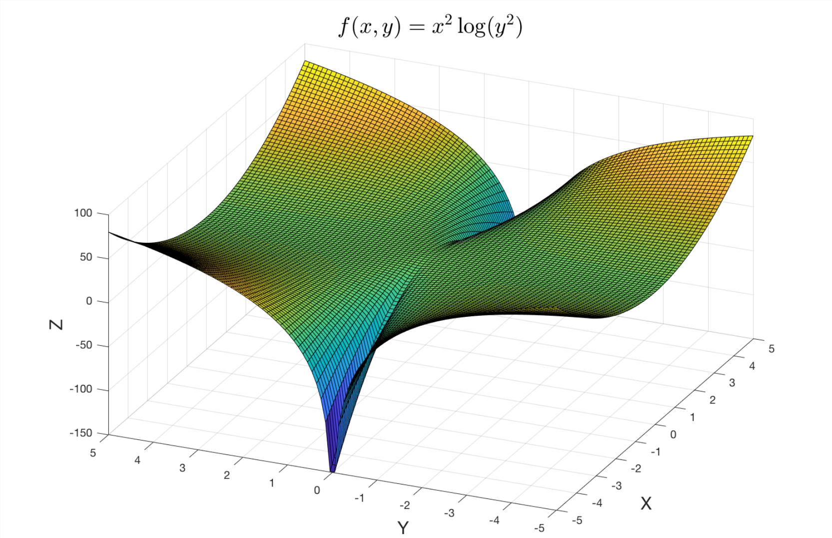
Code Demonstration

- Troubleshooting: Input guess of $(0, 0)$ returns (nan, nan)
- Need to include the `complex` library and pick only real component of returned value
- Problem remains; Input $(2, 3)$, Minimum $\left(7.0 \times 10^{33}, 9.5 \times 10^{33}\right)$

Code Demonstration

- Let's find the minimum of the function

$$f(x,y) = x^2 \log\left(y^2\right)$$



Code Demonstration

- The same issue of a returned minimum being (nan, nan) persists
- The input function $f(x,y) = x^2 \log\left(y^2\right)$ is real-valued $\forall\ x, y \in \mathbb{R}$
- $\Rightarrow$ This algorithm is not useful for functions that have logarithms in either the function or the derivative!

References

[1] W. Press, S. Teukolsky, W. Vetterling, & B. Flannery, *Numerical Recipes: The Art of Scientific Computing*, 2007, Cambridge University Press.