# Numerical Optimization

Lecture Notes #16

Calculating Derivatives — Finite Differencing

Peter Blomgren,
⟨blomgren.peter@gmail.com⟩

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

http://terminus.sdsu.edu/

Fall 2018





## Outline







## Derivatives Needed!!!

As we have seen (and will see), algorithms for nonlinear optimization (and nonlinear equations) require knowledge of derivatives:

| Nonlinear Optimization | Nonlinear Equations |
|---|---|
| **Gradient**, vector, 1st order<br>**Hessian**, matrix, 2nd order | **Jacobian**, matrix, 1st order |

Often it is quite trivial to provide the code which computes those derivatives, but in some cases the analytic expressions for the derivatives are not available and/or not practical to evaluate.

In those cases we need some other way to compute or **approximate** the derivatives.





## Finite Differences — The Return of Taylor's Theorem

We can get an approximation of the gradient $\nabla f(\bar{\mathbf{x}})$ by evaluating the objective $f$ at $(\mathbf{n+1})$ points, using the

Forward Difference Formula

$$\frac{\partial f(\bar{\mathbf{x}})}{\partial x_i} \approx \frac{f(\bar{\mathbf{x}}+\epsilon\bar{\mathbf{e}}_i)-f(\bar{\mathbf{x}})}{\epsilon}, \quad i=1,2,\ldots,n,$$

where $\bar{\mathbf{e}}_i$ is the $i$th unit vector, and $\epsilon > 0$ is small.

If $f$ is twice continuously differentiable, then by **Taylor's Theorem**

$$f(\bar{\mathbf{x}}+\bar{\mathbf{p}}) = f(\bar{\mathbf{x}}) + \nabla f(\bar{\mathbf{x}})^T\bar{\mathbf{p}} + \frac{1}{2}\bar{\mathbf{p}}^T\nabla^2 f(\bar{\mathbf{x}}+t\bar{\mathbf{p}})\bar{\mathbf{p}}, \quad t\in(0,1),$$

with $\bar{\mathbf{p}} = \epsilon\bar{\mathbf{e}}_i$, *i.e.*

$$f(\bar{\mathbf{x}}+\epsilon\bar{\mathbf{e}}_i) = f(\bar{\mathbf{x}}) + \epsilon\nabla f(\bar{\mathbf{x}})^T\bar{\mathbf{e}}_i + \frac{1}{2}\epsilon^2\bar{\mathbf{e}}_i^T\nabla^2 f(\bar{\mathbf{x}}+t\epsilon\bar{\mathbf{e}}_i)\bar{\mathbf{e}}_i, \quad i=1,2,\ldots,n.$$

## Forward Differences

Building the Gradient

With a bit of re-arrangement we see

$$\underbrace{\nabla f(\bar{\mathbf{x}})^T \bar{\mathbf{e}}_i}_{\frac{\partial f(\bar{\mathbf{x}})}{\partial x_i}} = \underbrace{\frac{f(\bar{\mathbf{x}} + \epsilon \bar{\mathbf{e}}_i) - f(\bar{\mathbf{x}})}{\epsilon}}_{\textbf{Finite Difference Approximation}} - \underbrace{\frac{1}{2}\epsilon\, \bar{\mathbf{e}}_i^T \nabla^2 f(\bar{\mathbf{x}} + t\epsilon \bar{\mathbf{e}}_i)\bar{\mathbf{e}}_i}_{\textbf{Approximation Error}}$$

If the Hessian $\nabla^2 f(\bar{\mathbf{x}})$ is bounded, *i.e.* $\|\nabla^2 f(\bar{\mathbf{x}})\| \le L_c$, then we have

$$\frac{\partial f(\bar{\mathbf{x}})}{\partial x_i} \approx \frac{f(\bar{\mathbf{x}} + \epsilon \bar{\mathbf{e}}_i) - f(\bar{\mathbf{x}})}{\epsilon},$$

where the approximation error is bounded by

$$\frac{\epsilon L_c}{2}.$$

Since the error is proportional to $\epsilon$, this is a **first-order approximation**.

San Diego State University





## Smaller is Better... Until it isn't



Ponder... The Silly Example™ —

```
g    = @(x) sin(cos(x));
FD_f = @(f,x,h) (f(x+h)-f(x))/h;

syms x
dg_ = diff(g(x),x);
dg = matlabFunction(dg_);
clear x

h  = 1;
x0 = pi/exp(1);
fprintf('--------------------------\n')
fprintf('h           (f(x+h)-f(x))/h\n')
fprintf('--------------------------\n')
for k = 1:20
    h = h / 10;
    fprintf('%6.2e    %15.8e\n',h, FD_f(g,x0,h))
end
fprintf('--------------------------\n')
fprintf('Exact:      %15.8e\n',dg(x0))
fprintf('--------------------------\n')
```

San Diego State University





## Smaller is Better... Until it isn't



```
--------------------------
h           (f(x+h)-f(x))/h
--------------------------
1.00e-01    -8.74662253e-01
1.00e-02    -8.45167500e-01
1.00e-03    -8.42038209e-01
1.00e-04    -8.41723614e-01
1.00e-05    -8.41692138e-01
1.00e-06    -8.41688990e-01
1.00e-07    -8.41688676e-01
1.00e-08    -8.41688641e-01   ⇐ Best approximation
1.00e-09    -8.41688663e-01
1.00e-10    -8.41688386e-01
1.00e-11    -8.41687831e-01
1.00e-12    -8.41771097e-01
1.00e-13    -8.40993941e-01
1.00e-14    -8.43769499e-01
1.00e-15    -8.88178420e-01
1.00e-16     0.00000000e+00
1.00e-17     0.00000000e+00
1.00e-18     0.00000000e+00
1.00e-19     0.00000000e+00
1.00e-20     0.00000000e+00
--------------------------
Exact:      -8.41688640e-01
--------------------------
```

San Diego State University





## Selecting $\epsilon$ — Machine Epsilon / Unit Roundoff



Clearly(?), the smaller the $\epsilon$ the smaller the error. How small can we set $\epsilon$ in *finite precision???*

Let $\epsilon_{\text{mach}}$ denote value for *machine epsilon*, a.k.a. **unit roundoff**, it is essentially the largest value for which

$$((1.0 + \epsilon_{\text{mach}}) - 1.0) = 0, \quad \textbf{in finite precision}$$

$\epsilon_{\text{mach}} \approx 10^{-16}$ in double-precision arithmetic (IEEE 64-bit floating point: "C" `double`, and Matlab internals on typical Intel-based systems.)

$\epsilon_{\text{mach}}$ is a measure of how well (or badly) we can represent any number in finite precision, and in extension a measure of the (best case) quality of every computation.

San Diego State University

## Selecting $\epsilon$



If $L_f$ is a bound on the value of $f(\bar{\mathbf{x}})$, *i.e.* $|f(\bar{\mathbf{x}})| \leq L_f$, then in finite precision we have

$$\begin{aligned} \|\texttt{computed}(f(\bar{\mathbf{x}})) - f(\bar{\mathbf{x}})\| &\leq \epsilon_{\text{mach}} L_f \\ \|\texttt{computed}(f(\bar{\mathbf{x}} + \epsilon\bar{\mathbf{e}}_i)) - f(\bar{\mathbf{x}} + \epsilon\bar{\mathbf{e}}_i)\| &\leq \epsilon_{\text{mach}} L_f. \end{aligned}$$

Now, if we recall our finite difference approximation

$$\frac{\partial f(\bar{\mathbf{x}})}{\partial x_i} \approx \frac{f(\bar{\mathbf{x}} + \epsilon\bar{\mathbf{e}}_i) - f(\bar{\mathbf{x}})}{\epsilon} + \texttt{error}(\epsilon),$$

we find that the total error is

$$\texttt{error}(\epsilon) \sim \underbrace{\frac{2\epsilon_{\text{mach}} L_f}{\epsilon}}_{\text{Floating Point Error}} + \underbrace{\frac{\epsilon L_c}{2}}_{\text{Approximation Error}}.$$





## Selecting $\epsilon$



Now,

$$\frac{d}{d\epsilon}\texttt{error}(\epsilon) \sim -\frac{2\epsilon_{\text{mach}} L_f}{\epsilon^2} + \frac{L_c}{2} = 0 \quad \Rightarrow \quad \epsilon^{\mathbf{2}} = \frac{\mathbf{4}\epsilon_{\text{mach}} \mathbf{L_f}}{\mathbf{L_c}},$$

gives us the optimal value for epsilon. Since $L_f$ and $L_c$ are unknown in general, most software packages tend to select

$$\epsilon^* = \sqrt{\epsilon_{\text{mach}}},$$

which is close to optimal in most cases.

Hence, the error in the approximated gradient is

$$\texttt{error}(\epsilon^*) \sim 2L_f\sqrt{\epsilon_{\text{mach}}} + \frac{L_c}{2}\sqrt{\epsilon_{\text{mach}}} \sim \mathcal{O}\left(\sqrt{\epsilon_{\text{mach}}}\right).$$





## Central Differences $\mathcal{O}\left(h^2\right)$ Accuracy



At twice the cost, we can get about 2.67 extra digits of precision in the finite difference approximation, by using **central differences**.

More Taylor expansions...

$$\begin{aligned} f(\bar{\mathbf{x}} + \epsilon\bar{\mathbf{e}}_i) &= f(\bar{\mathbf{x}}) + \epsilon\frac{\partial f}{\partial x_i} + \frac{1}{2}\epsilon^2\frac{\partial^2 f}{\partial x_i^2} + \mathcal{O}(\epsilon^3) \\ f(\bar{\mathbf{x}} - \epsilon\bar{\mathbf{e}}_i) &= f(\bar{\mathbf{x}}) - \epsilon\frac{\partial f}{\partial x_i} + \frac{1}{2}\epsilon^2\frac{\partial^2 f}{\partial x_i^2} + \mathcal{O}(\epsilon^3) \\ \hline f(\bar{\mathbf{x}} + \epsilon\bar{\mathbf{e}}_i) - f(\bar{\mathbf{x}} - \epsilon\bar{\mathbf{e}}_i) &= 2\epsilon\frac{\partial f}{\partial x_i} + \frac{\epsilon^3}{3}\frac{\partial^3 f}{\partial x_i^3} + \mathcal{O}(\epsilon^5). \end{aligned}$$

We get

**Central Difference Formula, with Error Term**

$$\frac{\partial f(\bar{\mathbf{x}})}{\partial x_i} = \frac{f(\bar{\mathbf{x}} + \epsilon\bar{\mathbf{e}}_i) - f(\bar{\mathbf{x}} - \epsilon\bar{\mathbf{e}}_i)}{2\epsilon} + \frac{\epsilon^2}{6}\frac{\partial^3 f}{\partial x_i^3} + \mathcal{O}(\epsilon^4).$$





## Central Differences $\mathcal{O}\left(h^2\right)$ Accuracy



Now, if we have a bound on the third derivative(s)

$$\left|\frac{\partial^3 f}{\partial x_i^3}\right| \leq L_J,$$

we can derive an optimal $\epsilon$:

$$\texttt{error}(\epsilon) \sim \frac{\epsilon_{\text{mach}} L_f}{\epsilon} + \frac{\epsilon^2 L_J}{6}.$$

$$\frac{d}{d\epsilon}\texttt{error}(\epsilon) \sim -\frac{\epsilon_{\text{mach}} L_f}{\epsilon^2} + \frac{\epsilon L_J}{3} = 0.$$

$$\Rightarrow \epsilon^* \sim \sqrt[3]{\frac{3\epsilon_{\text{mach}} L_f}{L_J}} \sim \sqrt[3]{\epsilon_{\text{mach}}} \quad \Rightarrow \quad \texttt{error} \sim \mathcal{O}\left(\epsilon_{\text{mach}}^{2/3}\right).$$

## Smaller is Better... Until it isn't — Redux



The Silly Example, now with central differences:

```
g    = @(x) sin(cos(x));
CD_f = @(f,x,h) (f(x+h)-f(x-h))/(2*h);

syms x
dg_ = diff(g(x),x);
dg = matlabFunction(dg_);
clear x

h  = 1;
x0 = pi/exp(1);
fprintf('-----------------------------\n')
fprintf('h           (f(x+h)-f(x-h))/2h\n')
fprintf('-----------------------------\n')
for k = 1:20
    h = h / 10;
    fprintf('%6.2e   %18.11e\n',h, CD_f(g,x0,h))
end
fprintf('-----------------------------\n')
fprintf('Exact:      %18.11e\n',dg(x0))
fprintf('-----------------------------\n')
```





## Smaller is Better... Until it isn't — Redux



```
------------------------------
h           (f(x+h)-f(x-h))/2h
------------------------------
1.00e-01    -8.39837686660e-01
1.00e-02    -8.41670105864e-01
1.00e-03    -8.41688455140e-01
1.00e-04    -8.41688638635e-01
1.00e-05    -8.41688640477e-01    ⇐ Best approximation
1.00e-06    -8.41688640424e-01
1.00e-07    -8.41688641007e-01
1.00e-08    -8.41688635456e-01
1.00e-09    -8.41688690967e-01
1.00e-10    -8.41688663211e-01
1.00e-11    -8.41690606102e-01
1.00e-12    -8.41771097271e-01
1.00e-13    -8.40993941154e-01
1.00e-14    -8.43769498715e-01
1.00e-15    -9.15933995316e-01
1.00e-16     0.00000000000e+00
1.00e-17     0.00000000000e+00
1.00e-18     0.00000000000e+00
1.00e-19     0.00000000000e+00
1.00e-20     0.00000000000e+00
------------------------------
Exact:      -8.41688640488e-01
------------------------------
```





## Approximating the Hessian — The Easy Case



**The easy case: Analytic Gradient given**

If the analytic gradient is known, then we can get an approximation of the Hessian by applying forward or central differencing to each element of the gradient vector in turn.

When the second derivatives exist and are Lipschitz continuous, **Taylor's theorem** says

$$\nabla f(\bar{\mathbf{x}}+\bar{\mathbf{p}}) = \nabla f(\bar{\mathbf{x}}) + \nabla^2 f(\bar{\mathbf{x}})\bar{\mathbf{p}} + \mathcal{O}(\|\bar{\mathbf{p}}\|^2).$$

Again, we let $\bar{\mathbf{p}} = \epsilon\bar{\mathbf{e}}_i$, $i = 1, 2, \dots, n$ and get

$$\nabla^2 f(\bar{\mathbf{x}})\bar{\mathbf{e}}_i \approx \frac{\nabla f(\bar{\mathbf{x}}+\epsilon\bar{\mathbf{e}}_i) - \nabla f(\bar{\mathbf{x}})}{\epsilon} + \mathcal{O}(\epsilon), \quad \textit{or}$$

$$\nabla^2 f(\bar{\mathbf{x}})\bar{\mathbf{e}}_i \approx \frac{\nabla f(\bar{\mathbf{x}}+\epsilon\bar{\mathbf{e}}_i) - \nabla f(\bar{\mathbf{x}}-\epsilon\bar{\mathbf{e}}_i)}{2\epsilon} + \mathcal{O}(\epsilon^2).$$





## Approximating the Hessian — Symmetrize



It is worth noting that this is a column-at-a-time process, which does not — due to numerical roundoff and approximation errors — necessarily give a symmetric Hessian.

It is often necessary to **symmetrize the result**

$$H_{\text{num}}^{\text{sym}} = \frac{1}{2}\left[H_{\text{num}} + H_{\text{num}}^T\right].$$

## Approximating the Hessian — Special Case 

**Special Case:** In Newton-CG methods we do not require full knowledge of the Hessian. Each iteration requires the Hessian-vector product $\nabla^2 f(\bar{\mathbf{x}})\bar{\mathbf{p}}$, where $\bar{\mathbf{p}}$ is the given search direction, this expression can be approximated

$$\nabla^2 f(\bar{\mathbf{x}})\bar{\mathbf{p}} \approx \frac{\nabla \mathbf{f}(\bar{\mathbf{x}}+\epsilon\bar{\mathbf{p}}) - \nabla f(\bar{\mathbf{x}}[-\epsilon\bar{\mathbf{p}}])}{[\mathbf{2}]\epsilon} + \mathcal{O}(\epsilon^{[2]})$$

This approximation is very cheap — only one **[two]** extra gradient evaluation**[s]** is **[are]** needed.





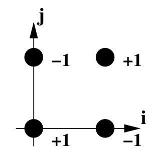
## Approximating the Hessian — Hard (Realistic) Case 

**The harder case: Analytic Gradient not given**

When the analytic gradient is not given we must use a finite difference formula using only function values to approximate the Hessian.

The first order **forward difference** approximation is given by

$$\frac{\partial^2 f(\bar{\mathbf{x}})}{\partial x_i \partial x_j} \approx \frac{f(\bar{\mathbf{x}}+\epsilon\bar{\mathbf{e}}_i+\epsilon\bar{\mathbf{e}}_j) - f(\bar{\mathbf{x}}+\epsilon\bar{\mathbf{e}}_i) - f(\bar{\mathbf{x}}+\epsilon\bar{\mathbf{e}}_j) + f(\bar{\mathbf{x}})}{\epsilon^2}$$








## Approximating the Hessian 

At a price of $\sim n^2$ additional function evaluations (an increase of 33%) we can use the second order **central difference approximation**

$$\frac{\partial^2 f(\bar{\mathbf{x}})}{\partial x_i \partial x_j} \approx \frac{f(\bar{\mathbf{x}}+\epsilon\bar{\mathbf{e}}_i+\epsilon\bar{\mathbf{e}}_j) - f(\bar{\mathbf{x}}+\epsilon\bar{\mathbf{e}}_i-\epsilon\bar{\mathbf{e}}_j) - f(\bar{\mathbf{x}}-\epsilon\bar{\mathbf{e}}_i+\epsilon\bar{\mathbf{e}}_j) + f(\bar{\mathbf{x}}-\epsilon\bar{\mathbf{e}}_i-\epsilon\bar{\mathbf{e}}_j)}{4\epsilon^2}$$

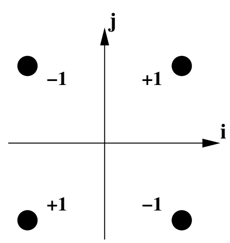



**Figure:** The second order 4-point central difference approximation stencil for $\frac{\partial^2 f(\bar{\mathbf{x}})}{\partial x_i \partial x_j}$ at the central point in the stencil — note that the value in that point is not part of the evaluation!





## Sparsity and Symmetry 

Now that we are paying $\sim 4$ function evaluations per entry in the Hessian matrix, it is worth taking sparsity and symmetry into account.

Ponder the extended Rosenbrock function:

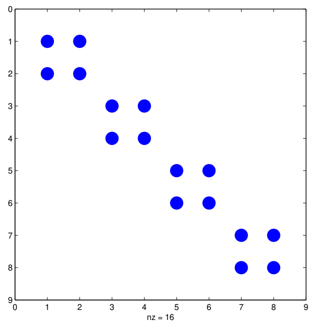
```
/* C/C++ code, why not?  */
double function_rosenbrock( int n, double *x )
{
  double f = 0.0;
  int    i;
  for( i=0; i<n/2; i++ )
     f += ( 10 * ( x[2*i+1] - x[2*i]*x[2*i] ) *
                 ( x[2*i+1] - x[2*i]*x[2*i] ) ) +
              ( 1 - x[2*i] ) * ( 1 - x[2*i] ) ;
  return(f);
}
```

Clearly, there is no "interaction" between coordinate-directions $\bar{\mathbf{e}}_i$ and $\bar{\mathbf{e}}_j$, where $|i-j|>1$.

## Sparsity and Symmetry



The fill-pattern of the Hessian of the extended Rosenbrock function consists of 2×2-diagonal blocks:




There are a lot of zero-entries in this Hessian. If somehow we have knowledge of the sparsity pattern, then we can exploit this by not computing/touching the zeros.





## Sparsity and Symmetry



By using the fact that the Hessian is symmetric, we can save about half of the work,

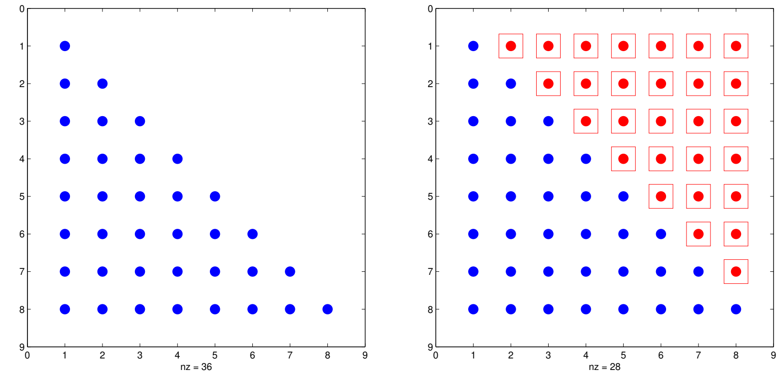



**Figure:** The entries to the left $H_{ij}$, $j \leq i$ must be computed, but using symmetry we can fill in the missing ones $H_{ij} = H_{ji}$, $j > i$.





## Index