CS 357: Numerical Methods

Floating Point

Eric Shaffer

Constrained Optimality

- If problem is constrained, only *feasible* directions are relevant
- For equality-constrained problem

$$\min f(\boldsymbol{x}) \quad \text{subject to} \quad \boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0}$$

where $f: \mathbb{R}^n \to \mathbb{R}$ and $\boldsymbol{g}: \mathbb{R}^n \to \mathbb{R}^m$, with $m \leq n$, necessary condition for feasible point $\boldsymbol{x}^*$ to be solution is that negative gradient of f lie in space spanned by constraint normals,

$$-\nabla f(\boldsymbol{x}^*) = \boldsymbol{J}_g^T(\boldsymbol{x}^*)\boldsymbol{\lambda}$$

where $\boldsymbol{J}_g$ is Jacobian matrix of $\boldsymbol{g}$, and $\boldsymbol{\lambda}$ is vector of *Lagrange multipliers*

- This condition says we cannot reduce objective function without violating constraints

Constrained Optimality

- *Lagrangian function* $\mathcal{L}\colon \mathbb{R}^{n+m} \to \mathbb{R}$, is defined by

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^T \boldsymbol{g}(\boldsymbol{x})$$

- Its gradient is given by

$$\nabla \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = \begin{bmatrix} \nabla f(\boldsymbol{x}) + \boldsymbol{J}_g^T(\boldsymbol{x})\boldsymbol{\lambda} \\ \boldsymbol{g}(\boldsymbol{x}) \end{bmatrix}$$

- Its Hessian is given by

$$\boldsymbol{H}_{\mathcal{L}}(\boldsymbol{x}, \boldsymbol{\lambda}) = \begin{bmatrix} \boldsymbol{B}(\boldsymbol{x}, \boldsymbol{\lambda}) & \boldsymbol{J}_g^T(\boldsymbol{x}) \\ \boldsymbol{J}_g(\boldsymbol{x}) & \boldsymbol{O} \end{bmatrix}$$

where

$$\boldsymbol{B}(\boldsymbol{x}, \boldsymbol{\lambda}) = \boldsymbol{H}_f(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i \boldsymbol{H}_{g_i}(\boldsymbol{x})$$

Constrained Optimality

- Together, necessary condition and feasibility imply critical point of Lagrangian function,

$$\nabla \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = \begin{bmatrix} \nabla f(\boldsymbol{x}) + \boldsymbol{J}_g^T(\boldsymbol{x})\boldsymbol{\lambda} \\ \boldsymbol{g}(\boldsymbol{x}) \end{bmatrix} = \boldsymbol{0}$$

- Hessian of Lagrangian is symmetric, but not positive definite, so critical point of $\mathcal{L}$ is saddle point rather than minimum or maximum
- Critical point $(\boldsymbol{x}^*, \boldsymbol{\lambda}^*)$ of $\mathcal{L}$ is constrained minimum of f if $\boldsymbol{B}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*)$ is positive definite on *null space* of $\boldsymbol{J}_g(\boldsymbol{x}^*)$
- If columns of $\boldsymbol{Z}$ form basis for null space, then test *projected* Hessian $\boldsymbol{Z}^T\boldsymbol{B}\boldsymbol{Z}$ for positive definiteness

Sequential Quadratic Programming

- For equality-constrained minimization problem

$$\min f(\boldsymbol{x}) \quad \text{subject to} \quad \boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0}$$

Newton

$x_{k+1} = x_k - H^{-1} \nabla f$

where $f\colon \mathbb{R}^n \to \mathbb{R}$ and $\boldsymbol{g}\colon \mathbb{R}^n \to \mathbb{R}^m$, with $m \le n$, we seek critical point of Lagrangian $\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^T \boldsymbol{g}(\boldsymbol{x})$

- Applying Newton's method to nonlinear system

$$\nabla \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = \begin{bmatrix} \nabla f(\boldsymbol{x}) + \boldsymbol{J}_g^T(\boldsymbol{x})\boldsymbol{\lambda} \\ \boldsymbol{g}(\boldsymbol{x}) \end{bmatrix} = \boldsymbol{0}$$

we obtain linear system

$$\begin{bmatrix} \boldsymbol{B}(\boldsymbol{x}, \boldsymbol{\lambda}) & \boldsymbol{J}_g^T(\boldsymbol{x}) \\ \boldsymbol{J}_g(\boldsymbol{x}) & \boldsymbol{O} \end{bmatrix} \begin{bmatrix} \boldsymbol{s} \\ \boldsymbol{\delta} \end{bmatrix} = - \begin{bmatrix} \nabla f(\boldsymbol{x}) + \boldsymbol{J}_g^T(\boldsymbol{x})\boldsymbol{\lambda} \\ \boldsymbol{g}(\boldsymbol{x}) \end{bmatrix}$$

for Newton step $(\boldsymbol{s}, \boldsymbol{\delta})$ in $(\boldsymbol{x}, \boldsymbol{\lambda})$ at each iteration

$$\nabla \mathcal{L} = \begin{bmatrix} \nabla f + J_g^T \lambda \\ g \end{bmatrix}$$

Constrained Optimization: Example

$f(x,y) = (x-2)^4 + 2(y-1)^2$ $\quad g(x,y) = x + 4y - 3 = 0$

$\mathcal{L}(x, y, \lambda) = f(x) + \lambda g(x)$ $\quad J_g = [1, 4]$

$$\boxed{\nabla \mathcal{L} = \begin{bmatrix} 4(x-2)^3 + \lambda \\ 4(y-1) + 4\lambda \\ x + 4y - 3 \end{bmatrix}}$$

$$H_{\mathcal{L}} = \begin{bmatrix} 12(x-2)^2, 0, 1 \\ 0, 4, 4 \\ 1, 4, 0 \end{bmatrix}$$

Constrained Optimization: Example

Constrained Optimization: Example

$$1.54 = \underbrace{+}_{\text{sign}} \underbrace{\left(1\times10^{0} + 5\times10^{-1} + 4\times10^{-2}\right)}_{\text{mantissa}} \times \underbrace{10^{0}}_{\text{exponent}}$$

Floating Point Numbers

- Floating-point number system is characterized by four integers

β	base or radix
p	precision
$[L, U]$	exponent range

- Number x is represented as

$$x = \pm \left(\underbrace{d_0 + \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \cdots + \frac{d_{p-1}}{\beta^{p-1}}} \right) \beta^E$$

where $0 \leq d_i \leq \beta - 1,\ i = 0, \ldots, p-1,$ and $L \leq E \leq U$

Floating Point Numbers

- Portions of floating-poing number designated as follows
  - *exponent*: E
  - *mantissa*: $d_0 d_1 \cdots d_{p-1}$
  - *fraction*: $d_1 d_2 \cdots d_{p-1}$
- Sign, exponent, and mantissa are stored in separate fixed-width *fields* of each floating-point *word*

IEEE 754

Parameters for typical floating-point systems

system	β	p	L	U
IEEE SP	2	24	-126	127
IEEE DP	2	53	-1022	1023

Normalization

- Floating-point system is *normalized* if leading digit d_0 is always nonzero unless number represented is zero
- In normalized systems, mantissa m of nonzero floating-point number always satisfies $1 \le m < \beta$
- Reasons for normalization
  - representation of each number unique
  - no digits wasted on leading zeros
  - leading bit need not be stored (in binary system)

Properties of Floating-Point Systems

- Floating-point number system is finite and discrete
- Total number of normalized floating-point numbers is

$$2(\beta - 1)\beta^{p-1}(U - L + 1) + 1$$

- Smallest positive normalized number: $\mathrm{UFL} = \beta^L$
- Largest floating-point number: $\mathrm{OFL} = \beta^{U+1}(1 - \beta^{-p})$
- Floating-point numbers equally spaced only between successive powers of β
- Not all real numbers exactly representable; those that are are called *machine numbers*

don't worry about this

$\beta = 2$

Example

$$\left.\begin{array}{l} 11.1 = 3.5 \\ 11.0 = 3 \\ 10.1 = 2.5 \\ 10.0 = 2 \end{array}\right\} 2^{1}$$

$$\left.\begin{array}{l} 1.11 = 1\,3/4 \\ 1.10 = 1\,1/2 \\ 1.01 = 1\,1/4 \\ 1.00 \quad 1 \end{array}\right\} 2^{0}$$

$$\left.\begin{array}{l} .111 = 7/8 \\ .110 = 3/4 \\ .101 = 5/8 \\ .100 = 1/2 \end{array}\right\} 2^{-1}$$

000

$m = d_0 d_1 d_2$

normalized

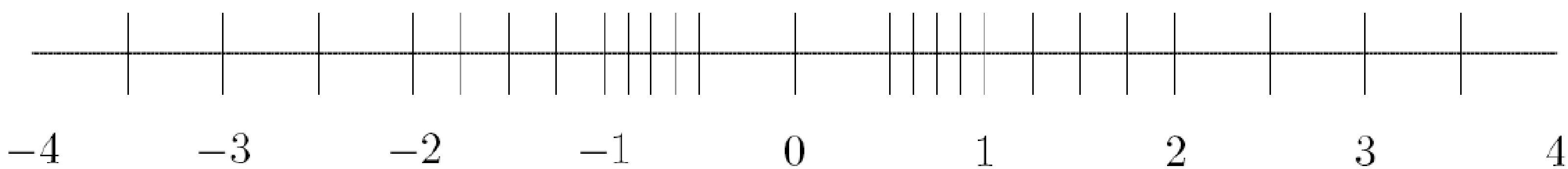
$1 \leq m < 2$

$d_0 = 1$

1 signbit

$E = [-1, 1]$

Example



- Tick marks indicate all 25 numbers in floating-point system having $\beta = 2$, $p = 3$, $L = -1$, and $U = 1$
  - $\mathrm{OFL} = (1.11)_2 \times 2^1 = (3.5)_{10}$
  - $\mathrm{UFL} = (1.00)_2 \times 2^{-1} = (0.5)_{10}$
- At sufficiently high magnification, all normalized floating-point systems look grainy and unequally spaced

Rounding Rules

- If real number x is not exactly representable, then it is approximated by "nearby" floating-point number $\mathrm{fl}(x)$
- This process is called *rounding*, and error introduced is called *rounding error*
- Two commonly used rounding rules
  - *chop*: truncate base-β expansion of x after $(p-1)$st digit; also called *round toward zero*
  - *round to nearest*: $\mathrm{fl}(x)$ is nearest floating-point number to x, using floating-point number whose last stored digit is even in case of tie; also called *round to even*
- Round to nearest is most accurate, and is default rounding rule in IEEE systems

Machine Precision

- Accuracy of floating-point system characterized by *unit roundoff* (or *machine precision* or *machine epsilon*) denoted by ϵ_{mach}
  - With rounding by chopping, $\epsilon_{\text{mach}} = \beta^{1-p}$
  - With rounding to nearest, $\epsilon_{\text{mach}} = \frac{1}{2}\beta^{1-p}$

don't worry about

- Alternative definition is smallest number ϵ such that $\text{fl}(1+\epsilon) > 1$

- Maximum relative error in representing real number x within range of floating-point system is given by

$$\boxed{\left|\frac{\text{fl}(x) - x}{x}\right| \leq \epsilon_{\text{mach}}}$$

Machine Precision

- For toy system illustrated earlier
  - $\epsilon_{\text{mach}} = (0.01)_2 = (0.25)_{10}$ with rounding by chopping
  - $\epsilon_{\text{mach}} = (0.001)_2 = (0.125)_{10}$ with rounding to nearest
- For IEEE floating-point systems
  - $\epsilon_{\text{mach}} = 2^{-24} \approx 10^{-7}$ in single precision
  - $\epsilon_{\text{mach}} = 2^{-53} \approx 10^{-16}$ in double precision
- So IEEE single and double precision systems have about 7 and 16 decimal digits of precision, respectively

Machine Precision

- Though both are "small," unit roundoff ϵ_{mach} should not be confused with underflow level UFL
- Unit roundoff ϵ_{mach} is determined by number of digits in *mantissa* of floating-point system, whereas underflow level UFL is determined by number of digits in *exponent* field
- In all *practical* floating-point systems,

$$0 < \text{UFL} < \epsilon_{\text{mach}} < \text{OFL}$$

Exceptional Values

- IEEE floating-point standard provides special values to indicate two exceptional situations
  - `Inf`, which stands for "infinity," results from dividing a finite number by zero, such as $1/0$
  - `NaN`, which stands for "not a number," results from undefined or indeterminate operations such as $0/0$, $0 * \texttt{Inf}$, or $\texttt{Inf}/\texttt{Inf}$
- `Inf` and `NaN` are implemented in IEEE arithmetic through special reserved values of exponent field

Floating-Point Arithmetic

- *Addition or subtraction*: Shifting of mantissa to make exponents match may cause loss of some digits of smaller number, possibly all of them
- *Multiplication*: Product of two p-digit mantissas contains up to $2p$ digits, so result may not be representable
- *Division*: Quotient of two p-digit mantissas may contain more than p digits, such as nonterminating binary expansion of $1/10$
- Result of floating-point arithmetic operation may differ from result of corresponding real arithmetic operation on same operands

Floating-Point Arithmetic

- Real result may also fail to be representable because its exponent is beyond available range
- Overflow is usually more serious than underflow because there is *no* good approximation to arbitrarily large magnitudes in floating-point system, whereas zero is often reasonable approximation for arbitrarily small magnitudes
- On many computer systems overflow is fatal, but an underflow may be silently set to zero

$$
\begin{array}{r}
1.92403 \times 10^{2} \\
- \; 1.92275 \times 10^{2} \\
\hline
1.28 \times 10^{-1}
\end{array}
$$

Cancellation

- Despite exactness of result, cancellation often implies serious loss of information
- Operands are often uncertain due to rounding or other previous errors, so relative uncertainty in difference may be large
- Example: if ϵ is positive floating-point number slightly smaller than ϵ_{mach}, then $(1+\epsilon)-(1-\epsilon)=1-1=0$ in floating-point arithmetic, which is correct for actual operands of final subtraction, but true result of overall computation, 2ϵ, has been completely lost
- Subtraction itself is not at fault: it merely signals loss of information that had already occurred

Cancellation

- Subtraction between two p-digit numbers having same sign and similar magnitudes yields result with *fewer* than p digits, so it is usually exactly representable
- Reason is that leading digits of two numbers *cancel* (i.e., their difference is zero)
- For example,

$$1.92403 \times 10^2 - 1.92275 \times 10^2 = 1.28000 \times 10^{-1}$$

which is correct, and exactly representable, but has only three significant digits

Cancellation

- Digits lost to cancellation are *most* significant, *leading* digits, whereas digits lost in rounding are *least* significant, *trailing* digits
- Because of this effect, it is generally bad idea to compute any small quantity as difference of large quantities, since rounding error is likely to dominate result
- For example, summing alternating series, such as

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

for $x < 0$, may give disastrous results due to catastrophic cancellation

$$\begin{array}{r} 1.01 \times 2^{0} \\ - \quad 1.00 \times 2^{0} \\ \hline 0.01 \end{array} = \boxed{2^{-2}}$$

$2^{0}\ 2^{-1}\ 2^{-2}$

$(d_0 \times 2^{0} + d_1 \times 2^{-1} \cdots\cdots) 2^{E}$

$d_0 . d_1 d_2 \quad 2^{1}$

$d_0 d_1 . d_2$

-1

$1 \times 2^{0} +$

0.10×2

0.11×2

1.0×2

1.01×2

Example: Standard Deviation

- Mean and standard deviation of sequence x_i, $i = 1, \ldots, n$, are given by

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \quad \text{and} \quad \sigma = \left[\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2\right]^{\frac{1}{2}}$$

- Mathematically equivalent formula

$$\sigma = \left[\frac{1}{n-1}\left(\sum_{i=1}^{n} x_i^2 - n\bar{x}^2\right)\right]^{\frac{1}{2}}$$

  avoids making two passes through data

- Single cancellation at end of one-pass formula is more damaging numerically than all cancellations in two-pass formula combined