目录

Introduction

这是一份学习文档

用于记录平时见遇到问题时查到的一些解决方法，所以没有明确的逻辑结构，而是分为几个板块，分别记录不同内容。

PartI Git

Chapter.1 Gitbook

记录了gitbook安装过程及一些常用命令。

参考GitBook 简明教程

Gitbook文档

Section.1 gitbook安装

1.安装

首先需要安装Node.js

在命令行输入 `node -v` 和 `npm -v` 确认安装完成。

命令行输入 `npm install gitbook-cli -g` 安装gitbook，如果出现问题，尝试使用 `cnpm install gitbook-cli -g` 来安装gitbook。

安装完成后，在指定目录下输入 `gitbook init` 初始化gitbook仓库，出现报错，提示为

> ../.../..../polyfills.js:287
>
> if(cb.apply(this, arguments))

打开指定文件polyfills.js后，找到该代码，其包含在函数statFix中，该函数似乎是为了解决旧版本返回uid+gid时候的数据类型的问题，新版本中不会出现该问题，所以注释使用的三行代码掉即可。（参考链接：https://www.jianshu.com/p/6221330b36ba）

> // fs.stat = statFix(fs.stat)
>
> // fs.fstat = statFix(fs.fstat)
>
> // fs.lstat = statFix(fs.lstat)

gitbook init 命令只是用来创建readme.md和SUMMARY.md两个文档，如果代码出错，直接创建连个代码，然后使用gitbook build即可。

1. 使用 `gitbook init` 初始化书籍目录
2. 使用 `gitbook serve` 编译书籍

gitbook fetch 3.2.3

gitbook fetch 2.6.7

等命令可以下载对应的版本

Section.2 gitbook 命令

gitbook init 命令只是用来创建readme.md和SUMMARY.md两个文档，如果代码出错，直接创建连个代码，然后使用gitbook build即可。

1. 使用 `gitbook init` 初始化书籍目录
2. 使用 `gitbook serve` 编译书籍

在文件修改过程中，每一次保存文件， `gitbook serve` 都会自动重新编译，所以可以持续通过浏览器来查看最新的书籍效果！

编译markdown文档，可以使用Typora_0.9（测试版本不收费，1.0后需要收费使用），或在VSCode中安装markdown组件。

在SUMMARY.md中添加相关章节后，执行 `gitbook init` ，会自动创建 `SUMMARY.md` 中的目录结构。

连续两个 `{` `{` 会导致build失败。

解决方法是在两个 `{` 之间添加一个空格，即 `{ {` ，即可解决问题。

Section.3 gitbook添加插件

通过添加插件可以使gitbook生成的电子书元素更加丰富。

在电子书根目录下创建book.json文件，

添加电子书的基础信息，

```
{
    "title" : "我的学习笔记",
    "author" : "litan",
    "description" : "日常工作中用到的一些技术总结",
    "language" : "zh-hans",
}
```

添加插件信息，

```
{
    "title" : "我的学习笔记",
    "author" : "litan",
    "description" : "日常工作中用到的一些技术总结",
    "language" : "zh-hans",
    "plugins" :[
        "katex",
        "expandable-chapters-small"
    ]
}
```

保存book.json文件后，命令行运行gitbook install，通过npm安装gitbook插件。

> 上段代码中\ katex插件可以使html正常显示TeX格式代码；\ expandable-chapters-small插件是使侧边栏伸缩的插件。
>
> 显示数学公式的插件已知有两个，分别是`mathjax`和`katex`， 公式插件详情参考网页。分别下载两种插件后发现，mathjax在下载时会报错（并没有详细查看报错原因和查找解决方案），而katex插件则可以正常下载和使用，所以比较推荐优先下载katex。

"back-to-top-button", 返回顶部
"lightbox 弹窗形式显示图片

Section.4 gitbook侧边栏导航问题

使用新下载的nodejs和npm安装gitbook `npm gitbook-cli -g` 会自动安装npm中最新版的gitbook（可能是3.2.3），该版本使用 `gitbook build` 生成的电子书侧边栏存在一些问题，以下是对应每个问题的解决方法。

1.侧边栏链接无法跳转

查找资料后，解决方法有两种，一是修改 _book/gitbook/theme.js文件，二是使用低版本gitbook。

a. 修改theme.js文件

打开theme.js文件，搜索 `if(m)for(n.handler&&` ，将 `if(m)` 修改为 `if(false)` ，即可。

这种方法的缺点是每次 `gitbook build` 后都需要修改该文件。

b.降低gitbook版本

网上很多博客都提到使用2.6.7版本的gitbook，但在下载和使用时总是会出现异常报错并且解决方法不容易搜索到。仅有一篇博客提到使用3.0.0版本，经过测试发现确实可行， `gitbook fetch 3.0.0` 下载该版本gitbook，没有异常报错情况出现，且 `gitbook build --gitbook=3.0.0` 指令可以正常完成。并且侧边栏可以正常跳转。

2.侧边栏父节点链接跳转异常

通过上述两种方法可以解决侧边栏不同页面之间无法跳转的问题，但使用时发现父节点信息（如chpter01/README.md等）仍有异常情况。

根据gitbook规则，在SUMMARY.md文档中，需要为每个父节点添加README.md文档，用于介绍父节点信息，但是在 `gitbook build` 后，打开网页点击父节点发现该链接为无法连接到README.md文档（如：./chapter01/这种情况）

缺图

经过实验，这种情况的解决方法是把**README.md文档改名**，具体的更改要求暂不清楚，但可能只要不只是README.md即可。（可以参考点击本文档父节点后跳转的链接）

Chapter.2 Git

记录了git常用命令和一些特殊情况的处理方法。

Section.1 git命令

1. git add . 添加至暂存区
2. git ls-files 查看add内容
3. git rm -r --cached 清除add在缓存里面的内容
4. git rm --cached <文件路径>（无"<"">"） 清除add在缓存里面的某个文件（不删除物理文件，仅将该文件从缓存中删除）
5. git commit -a -m "…" 提交改变的文件，-a提交所有改变，-m附加注释
6. git push origin master 将本地分支岁送到服务器上的master分支中，origin是服务器地址，master是自己的分支名
7. git push <远程主机名> <本地分支名>:<远程分支名> 将本地分支送到服务器上的远程分支中
8. git init --bare 初始化仓库，初始化终端仓库时需要加"--bare"
9. git pull origin master 将服务器的版本同步至本地，origin是服务器地址，本地分支名与远程分支名均为master
10. git pull <远程主机名> <远程分支名>:<本地分支名> 将服务器的远程分支版本同步至本地的分支中
11. git status 查看仓库与本地的状态
12. git diff +文件 查看文件的修改内容
13. git remote add origin /Z/guihua/share… 添加远程库的路径的"快捷方式"，origin = Z:/guihua/share…
14. git remote -v 查看远程库信息
15. git remote rm name 删除远程仓库
16. git remote add 使用指定字符串name来代替整个URL
17. git remote rename old new 更新仓库名，从old更新到new
18. git log 查看版本的历史记录
19. git log --oneline --graph --all 以图像（graph）的形式，简短（oneline）的显示所有（all）版本信息
20. git reset --hard HEAD^

git reset --hard 版本号 回溯到上个版本/指定版本（版本号使用git log可查询）

1. 首次提交代码时：

git config --global user.name "runoob"

git config --global user.email "runoob@email.com"

1. git branch 创建分支名
2. git branch 查看已存在分支和当前分支
3. git branch -d/D 删除本地分支（之前应checkout切换到其他分支）
4. git checkout 切换分支名
5. git checkout -b 创建并切换到该分支
6. git merge branch 将branch分支合并到当前分支中
7. git merge --abort 退出合并，不解决合并冲突

Section.2 git pull & clone

使用git命令下拉远程库。

选择创建的

pull

1. 首次下拉远程版本库时，需要初始化git库。

   `git init`

2. 添加远程库路径的“快捷方式”

   `git remote add origin /z/guihua/share/InSAR/LandSARGitRepostory`

   `origin` 为任意字符串，可根据情况修改

   origin后面输入远程库的真实路径

3. 下拉远程git库

   `git pull origin master`

   `git pull origin origin_branch:local_branch`

   `origin` 也可以使用真实路径替换

   对比本地分支和远程分支相同的情况，可以直接填写一个 `master` 即可，但若遇到远程库分支与本地分支不同的情况则需要输入 `branch_a:branch_b` 。

clone

使用clone的方式下拉远程库，方法大致与pull相同，但不能使用字符替换地址，必须使用真实路径

`git clone /z/guihua/share/InSAR/LandSARGitRepostory`

Chapter.3 MarkDown

Section.1 常规用法

Section.2 LaTeX语法

编号	大写	拼写	小写	拼写	发音
1	$\Alpha$	\Alpha	α	\alpha	a:lf
2	$\Beta$	\Beta	β	\beta	bet
3	Γ	\Gamma	γ	\gamma	ga:m
4	Δ	\Delta	δ	\delta	delt
5	$\Epsilon$	\Epsilon	ϵ	\epsilon	ep`silon
6	$\Zeta$	\Zeta	ζ	\zeta	zat
7	$\Eta$	\Eta	η	\eta	eit
8	Θ	\Theta	θ	\theta	θit
9	$\Iota$	\Iota	ι	\iota	aiot
10	$\Kappa$	\Kappa	κ	\kappa	kap
11	Λ	\Lambda	λ	\lambda	lambd
12	$\Mu$	\Mu	μ	\mu	mju
13	$\Nu$	\Nu	ν	\nu	nju
14	Ξ	\Xi	ξ	\xi	ksi
15	$\Omicron$	\Omicron	$\omicron$	\omicron	omik`ron
16	Π	\Pi	π	\pi	pai
17	$\Rho$	\Rho	ρ	\rho	rou
18	Σ	\Sigma	σ	\sigma	`sigma
19	$\Tau$	\Tau	τ	\tau	tau
20	Υ	\Upsilon	υ	\upsilon	jup`silon
21	Φ	\Phi	ϕ	\phi	fai
22	$\Chi$	\Chi	χ	\chi	phai
23	Ψ	\Psi	ψ	\psi	psai
24	Ω	\Omega	ω	\omega	o`miga

PartII Cpp

Chapter.1 Std

Section.1 LINUX & WIN32

头文件替换

library	windows	linux
io	#include	?
dirent	#include	#include

sprintf_s -> snprintf

```
char tag[256];
char* str;
#ifdef windows
        sprintf_s(tag,"<%s>",str);
#elif defined __linux__
        snprintf(tag,256,"<%s>",str);
#endif
```

fopen_s -> fopen

```
const char* charPath
FILE *pFile
#ifdef windows
fopen_s(&pFile,charPath,"rb");
#elif defined __linux__
pFile = fopen(charPath,"rb");
#endif
```

memset

linux下需要添加string.h （不是\）

_strlwr 平替（无函数）

```
char str[256];
 #ifdef windows
_strlwe(str);
#elif defined __linux__
for(char* ptr = str; *ptr; ptr++){
*ptr = tolower(*ptr);
}
#endif
```

_splitpath_s 平替（无函数）

```
1.SARImgIO.h中，只去后缀szExt的情况：
#include <stdio.h>
#include <string.h>

#ifdef windows
_splitpath_s(szFileName, szDri, szDir, szName, szExt);
#elif defined __linux__
std::string strFileName = szFileName;
std::string::size_type pos = strFileName.find_last_of(".");
strcpy(szExt, strFileName.substr(pos, strFileName.size()-1).c_str());
#endif
2.....（未完成）
```

_fseeki64 -> fseeko64

西电 孙浩南好像说这里函数计算结果不相同？

```
#ifdef windows
_fseeki64(file, (nSizeBOff+(nLWidth+i*imgWidth)*nSizeT), SEEK_SET);
#elif defined __linux__
_fseeko64(file, (nSizeBOff+(nLWidth+i*imgWidth)*nSizeT), SEEK_SET);
#endif
```

CPU数量

openmp需要使用

```
int intnCPUs;
#ifdef windows
#include<windows.h>
SYSTEM_INFO SystemInfo;
GetSystemInfo(&SystemInfo);
intnCPUs = SystemInfo.dwNumberOfProcessors;
#elif defined __linux__
#include <unistd.h>
#include <iostream>
intnCPUs = sysconf(_SC_NPROCESSORS_CONF);
#endif
intnCPUs *= 0.9;
```

Section.2 fstream

ofstream 写文档

```
#include<fstream>
ofstream temp(Filepath.txt);
temp<<……;
temp.close();
```

ifstream 读文档

```
ifstream ifs;
ifs.open(txtsrc);
if (!ifs.is_open())
{
std::cout << "文件打开失败" << endl;
system("pasue");
exit(0);
}

string str;
string Separator = string(",");
while (getline(ifs, str))
{
    vector<string> strVec;
    SplitString(str, strVec, Separator);
    ComparisonData x0;
    x0.jRan_real = atoi(strVec[0].c_str());
    x0.iAzi_real = atoi(strVec[1].c_str());
    x0.jRan_sim = atoi(strVec[2].c_str());
    x0.iAzi_sim = atoi(strVec[3].c_str());

    PointsCompareData.push_back(x0);
}
```

Section.3 string分割

参数说明

输入字符串str，

输出待分割的字符串数组strList，

输入分割符号split。

代码

```
#include <iostream>
#include <string>
#include <vector>

using std::vector;
using std::string;

void strSplit(string str, vector<string> &strList, string split){
    // strList.clear();
    std::string::size_type pos1, pos2;
    pos2 = str.find(split);
    pos1 = 0;
    while(std::string::npos != pos2){
        strList.push_back(str.substr(pos1,pos2-pos1));
        pos1 = pos2 + split.size();
        pos2 = str.find(split, pos1);
    }
    if (pos1 != str.length())
        strList.push_back(str.substr(pos1));
}
```

Section.4 转动的线

```
#include<Windows.h>
while (1)
{
    for (int i = 0; i < 12; i++)
    {
        printf("\b%c", "\\|/-\\|/-\\|/-"[i]);
        Sleep(150);
    }
}
```

sleep首字母大写,数字应该是以毫秒为单位 \b=backspace 退格键

16是4的3倍 对应的，下面要写三组“\|/-” 注意：不能使用中文符号 会发生错误

Chapter.2 GDAL

Chapter.3 Opencv

Chapter.4 TinyXML

Chapter.5 Eigen

Eigen库的使用说明

参考Eigen库官方文档

Section.1 基础操作预留1-20

Section.2 与Matlab命令对比

// A simple quickref for Eigen. Add anything that's missing.

// Main author: Keir Mierle

```
#include <Eigen/Dense>

Matrix<double, 3, 3> A;                // Fixed rows and cols. Same as Matrix3d.
Matrix<double, 3, Dynamic> B;          // Fixed rows, dynamic cols.
Matrix<double, Dynamic, Dynamic> C;    // Full dynamic. Same as MatrixXd.
Matrix<double, 3, 3, RowMajor> E;      // Row major; default is column-major.
Matrix3f P, Q, R;                      // 3x3 float matrix.
Vector3f x, y, z;                      // 3x1 float matrix.
RowVector3f a, b, c;                   // 1x3 float matrix.
VectorXd v;                            // Dynamic column vector of doubles
double s;
```

// Basic usage

矩阵/向量的尺寸、值

```
// Eigen          // Matlab           // comments
x.size()          // length(x)        // vector size
C.rows()          // size(C,1)        // number of rows
C.cols()          // size(C,2)        // number of columns
x(i)              // x(i+1)           // Matlab is 1-based
C(i,j)            // C(i+1,j+1)       //
```

矩阵重定义大小

```
A.resize(4, 4);   // Runtime error if assertions are on.
B.resize(4, 9);   // Runtime error if assertions are on.
A.resize(3, 3);   // Ok; size didn't change.
B.resize(3, 9);   // Ok; only dynamic cols changed.
```

矩阵填充

```
A << 1, 2, 3,     // Initialize A. The elements can also be
     4, 5, 6,     // matrices, which are stacked along cols
     7, 8, 9;     // and then the rows are stacked.
B << A, A, A;     // B is three horizontally stacked A's.
A.fill(10);       // Fill A with all 10's.
```

特殊矩阵构建

```
// Eigen                                  // Matlab
//单位矩阵
MatrixXd::Identity(rows,cols)             // eye(rows,cols)
C.setIdentity(rows,cols)                  // C = eye(rows,cols)
//全0矩阵
MatrixXd::Zero(rows,cols)                 // zeros(rows,cols)
C.setZero(rows,cols)                      // C = zeros(rows,cols)
//全1矩阵
MatrixXd::Ones(rows,cols)                 // ones(rows,cols)
C.setOnes(rows,cols)                      // C = ones(rows,cols)
//随机矩阵
MatrixXd::Random(rows,cols)               // rand(rows,cols)*2-1            // Matri
C.setRandom(rows,cols)                    // C = rand(rows,cols)*2-1
```

向量的等距分布

```
VectorXd::LinSpaced(size,low,high)        // linspace(low,high,size)'
v.setLinSpaced(size,low,high)             // v = linspace(low,high,size)'
VectorXi::LinSpaced(((hi-low)/step)+1,    // low:step:hi
                    low,low+step*(size-1))  //
```

// Matrix slicing and blocks. All expressions listed here are read/write.\ // Templated size versions are faster. Note that Matlab is 1-based (a size N\ // vector is x(1)...x(N)).\ / *PLEASE HELP US IMPROVING THIS SECTION* /

/ *Eigen 3.4 supports a much improved API for sub-matrices, including,* /

/ *slicing and indexing from arrays:* /

/ *http://eigen.tuxfamily.org/dox-devel/group__TutorialSlicingIndexing.html* /

矩阵/向量的区域选取（块）

```
Matrix3f P;
Vector3f x;
// Eigen                           // Matlab
x.head(n)                          // x(1:n)
x.head<n>()                        // x(1:n)
x.tail(n)                          // x(end - n + 1: end)
x.tail<n>()                        // x(end - n + 1: end)
x.segment(i, n)                    // x(i+1 : i+n)
x.segment<n>(i)                    // x(i+1 : i+n)
P.block(i, j, rows, cols)          // P(i+1 : i+rows, j+1 : j+cols)
P.block<rows, cols>(i, j)          // P(i+1 : i+rows, j+1 : j+cols)
P.row(i)                           // P(i+1, :)
P.col(j)                           // P(:, j+1)
P.leftCols<cols>()                 // P(:, 1:cols)
P.leftCols(cols)                   // P(:, 1:cols)
P.middleCols<cols>(j)              // P(:, j+1:j+cols)
P.middleCols(j, cols)              // P(:, j+1:j+cols)
P.rightCols<cols>()                // P(:, end-cols+1:end)
P.rightCols(cols)                  // P(:, end-cols+1:end)
P.topRows<rows>()                  // P(1:rows, :)
P.topRows(rows)                    // P(1:rows, :)
P.middleRows<rows>(i)              // P(i+1:i+rows, :)
P.middleRows(i, rows)              // P(i+1:i+rows, :)
P.bottomRows<rows>()               // P(end-rows+1:end, :)
P.bottomRows(rows)                 // P(end-rows+1:end, :)
P.topLeftCorner(rows, cols)        // P(1:rows, 1:cols)
P.topRightCorner(rows, cols)       // P(1:rows, end-cols+1:end)
P.bottomLeftCorner(rows, cols)     // P(end-rows+1:end, 1:cols)
P.bottomRightCorner(rows, cols)    // P(end-rows+1:end, end-cols+1:end)
P.topLeftCorner<rows,cols>()       // P(1:rows, 1:cols)
P.topRightCorner<rows,cols>()      // P(1:rows, end-cols+1:end)
P.bottomLeftCorner<rows,cols>()    // P(end-rows+1:end, 1:cols)
P.bottomRightCorner<rows,cols>()   // P(end-rows+1:end, end-cols+1:end)
```

// Of particular note is Eigen's swap function which is highly optimized. // Eigen // Matlab R.row(i) = P.col(j); // R(i, :) = P(:, j) R.col(j1).swap(mat1.col(j2)); // R(:, [j1 j2]) = R(:, [j2, j1])

// Views, transpose, etc;

/ *PLEASE HELP US IMPROVING THIS SECTION* /

/ *Eigen 3.4 supports a new API for reshaping:* /

/ *http://eigen.tuxfamily.org/dox-devel/group__TutorialReshape.html* /

矩阵的转置、共轭等

```
// Eigen                           // Matlab
R.adjoint()                        // R'
R.transpose()                      // R.' or conj(R')       // Read-write
R.diagonal()                       // diag(R)               // Read-write
x.asDiagonal()                     // diag(x)
R.transpose().colwise().reverse()  // rot90(R)              // Read-write
R.rowwise().reverse()              // fliplr(R)
R.colwise().reverse()              // flipud(R)
R.replicate(i,j)                   // repmat(P,i,j)
```

矩阵运算

```
// All the same as Matlab, but matlab doesn't have *= style operators.
// Matrix-vector.  Matrix-matrix.   Matrix-scalar.
y  = M*x;          R  = P*Q;        R  = P*s;
a  = b*M;          R  = P - Q;      R  = s*P;
a *= M;            R  = P + Q;      R  = P/s;
                   R *= Q;          R  = s*P;
                   R += Q;          R *= s;
                   R -= Q;          R /= s;

// Vectorized operations on each element independently
// Eigen                       // Matlab
R = P.cwiseProduct(Q);         // R = P .* Q
R = P.array() * s.array();     // R = P .* s
R = P.cwiseQuotient(Q);        // R = P ./ Q
R = P.array() / Q.array();     // R = P ./ Q
R = P.array() + s.array();     // R = P + s
R = P.array() - s.array();     // R = P - s
R.array() += s;                // R = R + s
R.array() -= s;                // R = R - s
R.array() < Q.array();         // R < Q
R.array() <= Q.array();        // R <= Q
R.cwiseInverse();              // 1 ./ P
R.array().inverse();           // 1 ./ P
R.array().sin()                // sin(P)
R.array().cos()                // cos(P)
R.array().pow(s)               // P .^ s
R.array().square()             // P .^ 2
R.array().cube()               // P .^ 3
R.cwiseSqrt()                  // sqrt(P)
R.array().sqrt()               // sqrt(P)
R.array().exp()                // exp(P)
R.array().log()                // log(P)
R.cwiseMax(P)                  // max(R, P)
R.array().max(P.array())       // max(R, P)
R.cwiseMin(P)                  // min(R, P)
R.array().min(P.array())       // min(R, P)
R.cwiseAbs()                   // abs(P)
R.array().abs()                // abs(P)
R.cwiseAbs2()                  // abs(P.^2)
R.array().abs2()               // abs(P.^2)
(R.array() < s).select(P,Q );  // (R < s ? P : Q)
R = (Q.array()==0).select(P,R) // R(Q==0) = P(Q==0)
R = P.unaryExpr(ptr_fun(func)) // R = arrayfun(func, P)   // with: scalar func(const s
```

矩阵的数值操作

```
// Reductions.
int r, c;
// Eigen                  // Matlab
R.minCoeff()              // min(R(:))
R.maxCoeff()              // max(R(:))
s = R.minCoeff(&r, &c)    // [s, i] = min(R(:)); [r, c] = ind2sub(size(R), i);
s = R.maxCoeff(&r, &c)    // [s, i] = max(R(:)); [r, c] = ind2sub(size(R), i);
R.sum()                   // sum(R(:))
R.colwise().sum()         // sum(R)
R.rowwise().sum()         // sum(R, 2) or sum(R')'
R.prod()                  // prod(R(:))
R.colwise().prod()        // prod(R)
R.rowwise().prod()        // prod(R, 2) or prod(R')'
R.trace()                 // trace(R)
R.all()                   // all(R(:))
R.colwise().all()         // all(R)
R.rowwise().all()         // all(R, 2)
R.any()                   // any(R(:))
R.colwise().any()         // any(R)
R.rowwise().any()         // any(R, 2)
```

点积、范数等

```
// Dot products, norms, etc.
// Eigen                  // Matlab
x.norm()                  // norm(x).    Note that norm(R) doesn't work in Eigen.
x.squaredNorm()           // dot(x, x)   Note the equivalence is not true for complex
x.dot(y)                  // dot(x, y)
x.cross(y)                // cross(x, y) Requires #include <Eigen/Geometry>
```

格式转换

```
//// Type conversion
// Eigen                  // Matlab
A.cast<double>();         // double(A)
A.cast<float>();          // single(A)
A.cast<int>();            // int32(A)
A.real();                 // real(A)
A.imag();                 // imag(A)
// if the original type equals destination type, no work is done
```

```
// Note that for most operations Eigen requires all operands to have the same type:
MatrixXf F = MatrixXf::Zero(3,3);
A += F;                // illegal in Eigen. In Matlab A = A+F is allowed
A += F.cast<double>(); // F converted to double and then added (generally, conversion
```

数组与**Eigen**矩阵的关联

```
// Eigen can map existing memory into Eigen matrices.
float array[3];
Vector3f::Map(array).fill(10);            // create a temporary Map over array and set
int data[4] = {1, 2, 3, 4};
Matrix2i mat2x2(data);                    // copies data into mat2x2
Matrix2i::Map(data) = 2*mat2x2;           // overwrite elements of data with 2*mat2x2
MatrixXi::Map(data, 2, 2) += mat2x2;      // adds mat2x2 to elements of data (alternat
```

```
// Solve Ax = b. Result stored in x. Matlab: x = A \ b.
x = A.ldlt().solve(b));  // A sym. p.s.d.    #include <Eigen/Cholesky>
x = A.llt() .solve(b));  // A sym. p.d.      #include <Eigen/Cholesky>
x = A.lu()  .solve(b));  // Stable and fast. #include <Eigen/LU>
x = A.qr()  .solve(b));  // No pivoting.     #include <Eigen/QR>
x = A.svd() .solve(b));  // Stable, slowest. #include <Eigen/SVD>
// .ldlt() -> .matrixL() and .matrixD()
// .llt()  -> .matrixL()
// .lu()   -> .matrixL() and .matrixU()
// .qr()   -> .matrixQ() and .matrixR()
// .svd()  -> .matrixU(), .singularValues(), and .matrixV()
```

特征值问题

```
// Eigenvalue problems
// Eigen                          // Matlab
A.eigenvalues();                  // eig(A);
EigenSolver<Matrix3d> eig(A);     // [vec val] = eig(A)
eig.eigenvalues();                // diag(val)
eig.eigenvectors();               // vec
// For self-adjoint matrices use SelfAdjointEigenSolver<>
```

Section.21 线性问题预留21-40

线性最小二乘

解决密集矩阵的线性最小二乘问题

$Ax = b$

Eigen文档中提供了三种方法，分别是SVD分解，QR分解和正规方程三种方式。其中，SVD分解通常是最准确但也是最慢的，正规方程最快但最不准确，QR介于两者之间。

SVD分解 (SVD decomposition)

```
#include <iostream>
#include <Eigen/Dense>

using namespace std;
using namespace Eigen;

int main()
{
   MatrixXf A = MatrixXf::Random(3, 2);
   cout << "Here is the matrix A:\n" << A << endl;
   VectorXf b = VectorXf::Random(3);
   cout << "Here is the right hand side b:\n" << b << endl;
   cout << "The least-squares solution is:\n"
        << A.bdcSvd(ComputeThinU | ComputeThinV).solve(b) << endl;
}

//output:
Here is the matrix A:
  0.68  0.597
-0.211  0.823
 0.566 -0.605
Here is the right hand side b:
 -0.33
 0.536
-0.444
The least-squares solution is:
-0.67
0.314
```

QR分解 (QR decomposition)

正规方程 (normal equations)

■$Ax = b$

$\Rightarrow A^T Ax = A^T b$

$\Rightarrow x = (A^T A)^{-1} A^T b$

Section.41 稀疏矩阵预留41-..

Chapter 99？ 无关的C++函数

Section.1 转动的线

```
#include<Windows.h>
while (1)
{
    for (int i = 0; i < 12; i++)
    {
        printf("\b%c", "\\|/-\\|/-\\|/-"[i]);
        Sleep(150);
    }
}

sleep首字母大写,数字应该是以毫秒为单位
\b=backspace 退格键
```

> 16是4的3倍
>
> 对应的，下面要写三组“\|/-”
>
> 注意：不能使用中文符号 会发生错误

Section.2 二进制float读取

```
bool slcRead(const char *src,float *fReal, float *fImag)
{
    ifstream inFile(src, ifstream::binary); //二进制读方式打开
    if (!inFile) {
        printf("ERROR: File open failed...\nFilepath is %s",string(src));
        return false;
    }
    bool bReal = true;
    int num = 0;
    unsigned int value2;
    while (inFile.read((char*)&value2, 4)) { //一直读到文件结束
        //高低位字节变换
        unsigned int idata1, idata2, idata3, idata4;
        idata1 = 255 & (value2 >> 24);
        idata2 = 255 & (value2 >> 16);
        idata3 = 255 & (value2 >> 8);
        idata4 = 255 & value2;
        char str[9];
        sprintf(str, "%02x%02x%02x%02x", idata4, idata3, idata2, idata1);
        str[8] = 0;
        float a;
        sscanf(str, "%x", &a);
        //分别存储到fReal和fImag中
        if (bReal) {
            fReal[num] = a;
            bReal = false;
            //printf("%d:real %f,\t", num2, a);
        }
        else {
            fImag[num] = a;
            bReal = true;
            num++;
            //printf("imag %f\n", a);
        }
    }
    return true;
}
```

Section.3 .mat读取

VS工程设置

> 1.VC++目录->包含目录 添加: MATLAB\R2010b\extern\include MATLAB\R2010b\extern\include\win64 ->库目录 添加：MATLAB\R2010b\extern\lib\win64\microsoft MATLAB\R2010b\extern\lib\win32\microsoft 2.C/C++->常规->附加包含目录 添加: MATLAB\R2010b\extern\include MATLAB\R2010b\extern\include\win64 3.链接器->输入->附加依赖库 添加: libmat.lib libmx.lib libmex.lib libeng.lib 由于所安装的matlab为64位，要调用其函数，需要将工程转换为X64 4.顶菜单->生成->配置管理器->平台：X64 5.计算机环境变量->path 添加：E:\DevTools\MATLAB\R2010b\extern\lib\win64\microsoft; E:\DevTools\MATLAB\R2010b\bin\win64;

mat读取代码

```
MATFile *pmatFile = NULL;
mxArray *pMxArray = NULL;
const char **dir;//元素名列表
int  ndir;//mat文件中的元素(矩阵、元胞)个数
pmatFile = matOpen("D:\\ICESat-2\\ZY303\\ATL03_20200906005252_ph_rmOutlier.mat", '
pMxArray = matGetVariable(pmatFile, "initA");


double *dst_rmOutlier;

dir = (const char **)matGetDir(pmatFile, &ndir);
pMxArray = matGetVariable(pmatFile, "dst_rmOutlier");

dst_rmOutlier = (double*)mxGetData(pMxArray);//默认为逐列表示 dst_rmOutlier(i,j)=ds
int M = mxGetM(pMxArray);//行
int N = mxGetN(pMxArray);//列
```

注：读取的矩阵是以列形式存储，矩阵的第i行第j列元素表示为 `array(i,j)=array[i+M*j]`

Section.4 for循环进度条

```
void processBar_ForFunction(const char* name,int i,int sum)
{
    if (i % (sum/100) == 0)
    {
        cout <<"\r"<<name<< "Process:";
        int tempNum = 100.0*i / sum;
        do
        {
            cout << ">";
            tempNum = tempNum - 2;
        } while (tempNum >= 0);
        cout << " " << int(100.0 * i / sum)+1 << "%";
    }
}
```

PartIII Qt

Chapter.1 LandApp

Section.1 LandApp / Core

Core::RegExp

RegExp属于core库，只有.h文件，记录了（#define）部分常用的正则表达式。

```
#define _Reg_Number "^[-|+]?[0-9]*$"
#define _Reg_Number_N "^\\d{n}$"
#define _Reg_Float "^[-|+]?([1-9]\\d*\\.\\d*|0\\.\\d*[1-9]\\d*|0?\\.0+|0)$"
#define _Reg_Telephone "^(12[0-9]|14[5|7]|15[0|1|2|3|5|6|7|8|9]|18[0|1|2|3|5|6|7|8|9])
```

Core::SysSpace

SysSpace属于core库，负责调用计算机中的标准路径（QStandardPaths类）。**solvePath()**函数负责调用InSAR_Console.exe所在路径（qapplicationDirPath+“/./plugins/InSAR_Console/InSAR_Console.exe”）;writeFile(QString, QStringList)将QStringList中存储的文本存储到QString路径下;docPath()函数负责调用操作手册路径。

QstandardPaths

configPath()里用到了QStandardPaths 顾名思义为标准路径，用于解决不同操作系统（跨平台）标准路径不同的问题，例如“我的文档”在不同操作系统中的目录位置

windows: C:/Users/Documents

Linux: ~/Documents

a.displayName()

QString QStandardPaths::displayName(QStandardPaths::StandardLocation type)

根据标准目录类型化，返回对应的本地名称，如果找不到，返回空QString。 例如指定type为DesktopLocation，则返回桌面名称；DocumentsLocation，则返回文档目录名称。

```
qDebug()<<QStandardPaths::displayName(QStandardPaths::DesktopLocation)

<<QStandardPaths::displayName(QStandardPaths::DocumentsLocation);
```

（可以自行实验查看输出值）

b.findExecutable()

QString QStandardPaths::findExecutable(const QString &executableName, const QStringList &paths = QStringList())

在指定的路径中查找名为executableName的可执行文件，如果paths为空，则在系统路径中查找。 系统路径指PATH环境变量的值。 如果存在，返回可执行文件的绝对文件路径，如果没有找到，则返回空字符串。

```
QStringList paths; paths << "D:/notepadd++/soft/Notepad++"; qDebug()
<<QStandardPaths::findExecutable("notepad++", paths);
```

（可以自行实验查看输出值）

c.writableLocation()

QStandardPaths::writableLocation(QStandardPaths::StandardLocation type)

用法与displayName()相似，返回对应的本地目录。如果找不到，返回空QString。

举例

```
// displayName qDebug() << "DesktopLocation: " <<
QStandardPaths::displayName(QStandardPaths::DesktopLocation); qDebug() <<
"DocumentsLocation: " <<
QStandardPaths::displayName(QStandardPaths::DocumentsLocation); qDebug() <<
"FontsLocation: " << QStandardPaths::displayName(QStandardPaths::FontsLocation);
qDebug() << "ApplicationsLocation: " <<
QStandardPaths::displayName(QStandardPaths::ApplicationsLocation); qDebug() <<
"MusicLocation: " << QStandardPaths::displayName(QStandardPaths::MusicLocation);
qDebug() << "MoviesLocation: " <<
QStandardPaths::displayName(QStandardPaths::MoviesLocation); qDebug() <<
"PicturesLocation: " << QStandardPaths::displayName(QStandardPaths::PicturesLocation);
qDebug() << "TempLocation: " <<
QStandardPaths::displayName(QStandardPaths::TempLocation); qDebug() << "HomeLocation:
" << QStandardPaths::displayName(QStandardPaths::HomeLocation); qDebug() <<
"DataLocation: " << QStandardPaths::displayName(QStandardPaths::DataLocation);
qDebug() << "CacheLocation: " <<
QStandardPaths::displayName(QStandardPaths::CacheLocation); qDebug() <<
"GenericCacheLocation: " <<
QStandardPaths::displayName(QStandardPaths::GenericCacheLocation); qDebug() <<
"GenericDataLocation: " <<
QStandardPaths::displayName(QStandardPaths::GenericDataLocation); qDebug() <<
"RuntimeLocation: " << QStandardPaths::displayName(QStandardPaths::RuntimeLocation);
qDebug() << "ConfigLocation: " <<
QStandardPaths::displayName(QStandardPaths::ConfigLocation); qDebug() <<
"DownloadLocation: " << QStandardPaths::displayName(QStandardPaths::DownloadLocation);
qDebug() << "GenericConfigLocation: " <<
QStandardPaths::displayName(QStandardPaths::GenericConfigLocation); qDebug() <<
"AppDataLocation: " << QStandardPaths::displayName(QStandardPaths::AppDataLocation);
qDebug() << "AppLocalDataLocation: " <<
QStandardPaths::displayName(QStandardPaths::AppLocalDataLocation); qDebug() <<
"AppConfigLocation: " <<
QStandardPaths::displayName(QStandardPaths::AppConfigLocation); // findExecutable
qDebug() << QStandardPaths::findExecutable("cmd.exe"); // writableLocation qDebug()
<< QStandardPaths::writableLocation(QStandardPaths::DownloadLocation); qDebug() <<
QStandardPaths::writableLocation(QStandardPaths::AppDataLocation);
```

原文链接：https://blog.csdn.net/luoshabugui/article/details/88012838

slovePath()

return QDir(qApp->applicationDirPath() +
"/./plugins/InSAR_Console/InSAR_Console.exe").absolutePath();

展示了qApp与InSAR_Console的相对位置关系

writeFile()

writeFile(QString filename, QStringList &text)

实际应用中，filename是文件路径，text是存储配置文件内容的QStringList。

……

Section.2 LandApp / IO, Plot, QXlsx & Util

IO

主要是文本的读read写write，以及拆分splitLine操作。

其中splitLine有两种方式，当输入一个参数QString时，函数会打开QString对应的文本文档，逐行拆分并输出QStringList；当输入两个参数QString、QString时，函数会以第二个QString作为分隔符，拆分第一个QString并输出QStringList。

Plot

调用qcustomplot函数（是一个基于Qt的画图和数据可视化C++控件）。
QCustomPlot 致力于提供美观的界面，高质量的2D画图、图画和图表，同时为实时数据可视化应用提供良好的解决方案。

QXlsx

是一个读取excel的，开源软件库。

Util

共包含三个函数、分别是：

1.QStringList util::readFromCSV(QString filename)：xt/csv数据的读取，逐行存储值QStringList并输出；

2.void util::LongitudeLatitudeToDMS(double value, int &deg, int &min, int &sec)：输入经纬度，转换为度分秒。

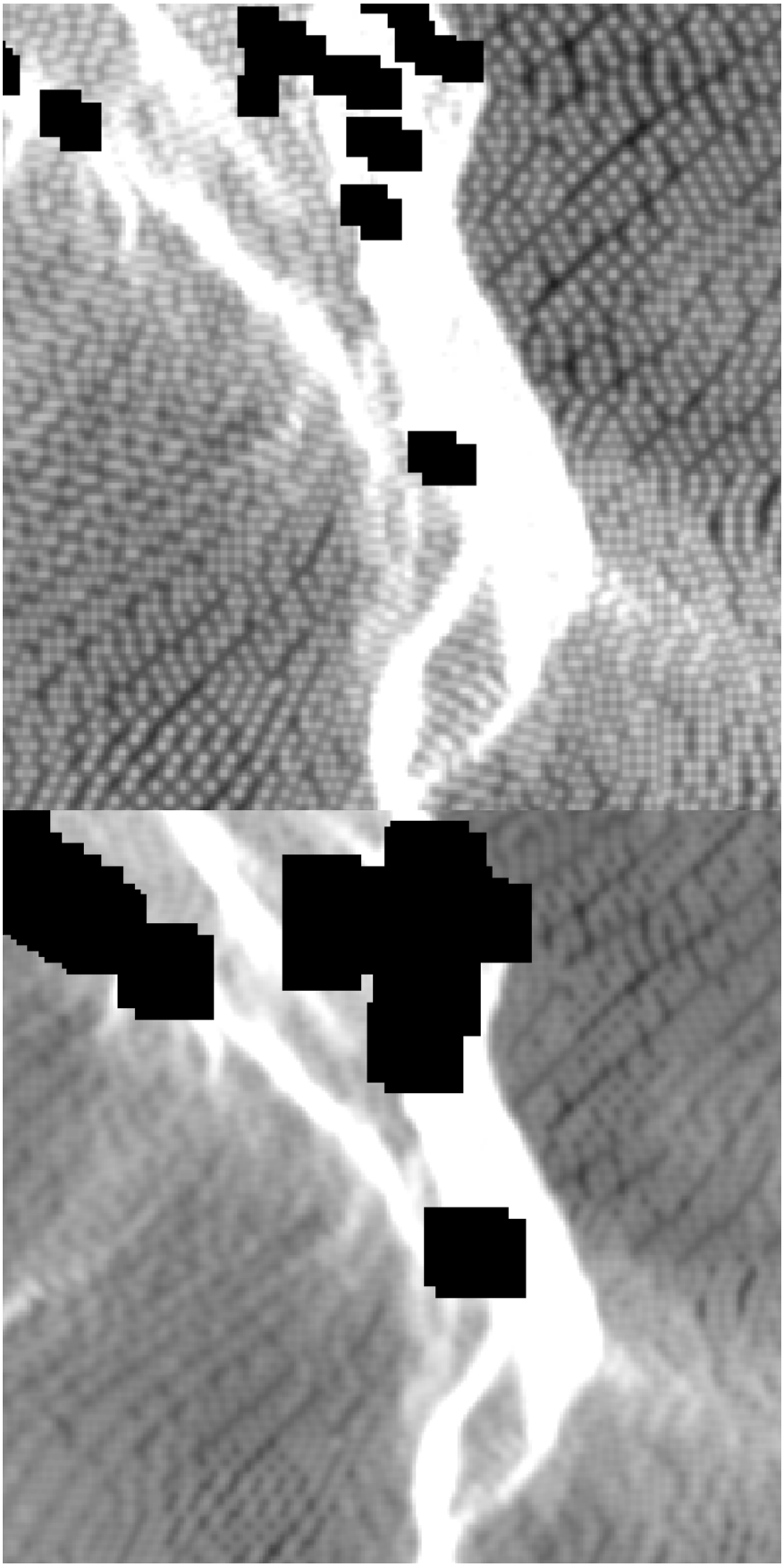
3.QString util::LongitudeLatitudeToDMS(double value)：输入经纬度，转换为度分秒，文本形式输出“%1° %2′ %3″ ”。

Chapter.3 LandSARWnd

Part IV RemoteSensing

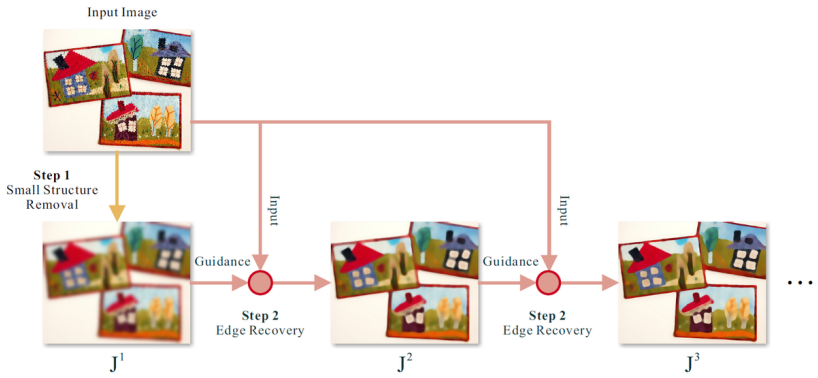
Chapter.1 图像处理

Section.1 RG Filter



pic.1 before filter. pic.2 after filter.

算法流程



Step1: Small Structure Removal

$$J_1(p)=\frac{1}{K_p}\sum_{q\in N(p)}\exp\left(-\frac{\|p-q\|^2}{2\sigma_s^2}\right)I(q)$$

$$K_p=\sum_{q\in N(p)}\exp\left(-\frac{\|p-q\|^2}{2\sigma_s^2}\right)$$

其中，p是待求点，q是p周围的所有点的集合，I是原图像，σ_s^2是标准偏差（standard deviation）自己设定。通过这一步可以消除尺度小于σ_s^2的结构。

（This filter completely removes structures whose scale is smaller than σ_s^2 as claimed in the scale space theory. It is implemented efficiently by separating kernels in perpendicular directions. Approximation by box filter is also feasible.）

Step2: Edge Recovery

$$J_{t+1}(p)=\frac{1}{K_p}\sum_{q\in N(p)}\exp\left(-\frac{\|p-q\|^2}{2\sigma_s^2}-\frac{\|J_t(p)-J_t(q)\|^2}{2\sigma_r^2}\right)I(q)$$

$$K_p=\sum_{q\in N(p)}\exp\left(-\frac{\|p-q\|^2}{2\sigma_s^2}-\frac{\|J_t(p)-J_t(q)\|^2}{2\sigma_r^2}\right)$$

其中，p是待求点，q是p周围的所有点的集合，t=1,2,...,n，I是原图像，σ_s^ 和 σ_r^ 分别控制时间权重和距离权重(σs and σr control the spatial and range weights respectively)。

C++代码

详见RG Filter代码

Chapter.2 坐标系转换

Section.1 UTM to WGS84 BLH

History

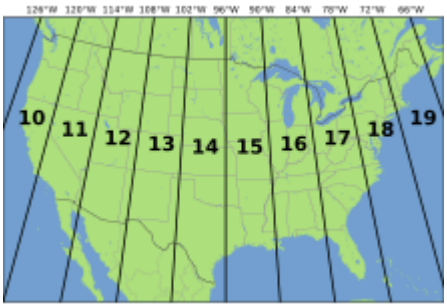
The National Oceanic and Atmospheric Administration (NOAA) website states the system to have been developed by the United States Army Corps of Engineers, starting in the early 1940s.[3] However, a series of aerial photos found in the Bundesarchiv-Militärarchiv (the military section of the German Federal Archives) apparently dating from 1943–1944 bear the inscription UTMREF followed by grid letters and digits, and projected according to the transverse Mercator,[4] a finding that would indicate that something called the UTM Reference system was developed in the 1942–43 time frame by the Wehrmacht. It was probably carried out by the Abteilung für Luftbildwesen (Department for Aerial Photography). From 1947 onward the US Army employed a very similar system, but with the now-standard 0.9996 scale factor at the central meridian as opposed to the German 1.0.[4] For areas within the contiguous United States the Clarke Ellipsoid of 1866[5] was used. For the remaining areas of Earth, including Hawaii, the International Ellipsoid[6] was used. The World Geodetic System WGS84 ellipsoid is now generally used to model the Earth in the UTM coordinate system, which means current UTM northing at a given point can differ up to 200 meters from the old. For different geographic regions, other datum systems can be used.

Prior to the development of the Universal Transverse Mercator coordinate system, several European nations demonstrated the utility of grid-based conformal maps by mapping their territory during the interwar period. Calculating the distance between two points on these maps could be performed more easily in the field (using the Pythagorean theorem) than was possible using the trigonometric formulas required under the graticule-based system of latitude and longitude. In the post-war years, these concepts were extended into the Universal Transverse Mercator/Universal Polar Stereographic (UTM/UPS) coordinate system, which is a global (or universal) system of grid-based maps.

The transverse Mercator projection is a variant of the Mercator projection, which was originally developed by the Flemish geographer and cartographer Gerardus Mercator, in 1570. This projection is conformal, which means it preserves angles and therefore shapes across small regions. However, it distorts distance and area.

Definitions[edit]

UTM zone[edit]



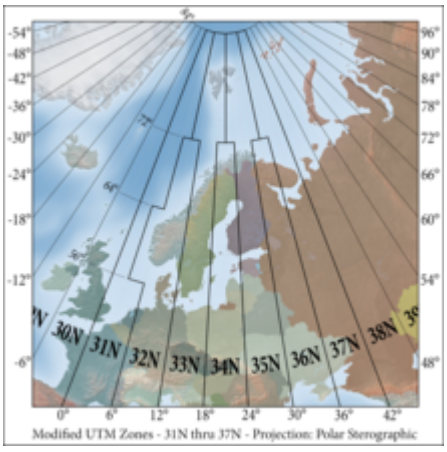
Simplified view of contiguous US UTM zones, projected with Lambert conformal conic.

The UTM system divides the Earth into 60 zones, each 6° of longitude in width. Zone 1 covers longitude 180° to 174° W; zone numbering increases eastward to zone 60, which covers longitude 174°E to 180°. The polar regions south of 80°S and north of 84°N are excluded.

Each of the 60 zones uses a transverse Mercator projection that can map a region of large north-south extent with low distortion. By using narrow zones of 6° of longitude (up to 668 km) in width, and reducing the scale) factor along the central meridian) to 0.9996 (a reduction of 1:2500), the amount of distortion is held below 1 part in 1,000 inside each zone. Distortion of scale increases to 1.0010 at the zone boundaries along the equator.

In each zone the scale factor of the central meridian reduces the diameter of the transverse cylinder to produce a secant projection with two standard lines, or lines of true scale, about 180 km on each side of, and about parallel to, the central meridian (Arc cos 0.9996 = 1.62° at the Equator). The scale is less than 1 inside the standard lines and greater than 1 outside them, but the overall distortion is minimized.

Overlapping grids[edit]



Universal Transverse Mercator (UTM) Grid Zones 31N thru 37N differ from the standard 6° wide by 84° zone for the northern hemisphere, in part to accommodate the southern half of the Kingdom of Norway. For more on its history, see Clifford J. Mugnier's article on Grids & Datums of The Kingdom of Norway that appeared in the October 1999 issue of PE&RS http://www.asprs.org/a/resources/grids/10-99-norway.pdf

Distortion of scale increases in each UTM zone as the boundaries between the UTM zones are approached. However, it is often convenient or necessary to measure a series of locations on a single grid when some are located in two adjacent zones. Around the boundaries of large scale maps (1:100,000 or larger) coordinates for both adjoining UTM zones are usually printed within a minimum distance of 40 km on either side of a zone boundary. Ideally, the coordinates of each position should be measured on the grid for the zone in which they are located, but because the scale factor is still relatively small near zone boundaries, it is possible to overlap measurements into an adjoining zone for some distance when necessary.

Latitude bands[edit]

Latitude bands are not a part of UTM, but rather a part of the military grid reference system (MGRS).[7] They are however sometimes used.

Latitude bands[edit]

Each zone is segmented into 20 latitude bands. Each latitude band is 8 degrees high, and is lettered starting from "C" at 80°S, increasing up the English alphabet until "X", omitting the letters "I" and "O" (because of their similarity to the numerals one and zero). The last latitude band, "X", is extended an extra 4 degrees, so it ends at 84°N latitude, thus covering the northernmost land on Earth.

Latitude bands "A" and "B" do exist, as do bands "Y" and "Z". They cover the western and eastern sides of the Antarctic and Arctic regions respectively. A convenient mnemonic to remember is that the letter "N" is the first letter in "northern hemisphere", so any letter coming before "N" in the alphabet is in the southern hemisphere, and any letter "N" or after is in the northern hemisphere.

Notation[edit]

The combination of a zone and a latitude band defines a grid zone. The zone is always written first, followed by the latitude band. For example, (see image, top right), a position in Toronto, Ontario, Canada, would find itself in zone 17 and latitude band "T", thus the full grid zone reference is "17T". The grid zones serve to delineate irregular UTM zone boundaries. They also are an integral part of the military grid reference system.

A note of caution: A method also is used that simply adds N or S following the zone number to indicate North or South hemisphere (the easting and northing coordinates along with the zone number supplying everything necessary to geolocate a position except which hemisphere). However, this method has caused some confusion since, for instance, "50S" can mean southern hemisphere but also *grid zone* "50S" in the northern hemisphere.[8] There are many possible ways to disambiguate between the two methods, two of which are demonstrated later in this article.

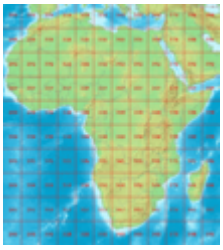
Exceptions[edit]

These grid zones are uniform over the globe, except in two areas. On the southwest coast of Norway, grid zone 32V (9° of longitude in width) is extended further west, and grid zone 31V (3° of longitude in width) is correspondingly

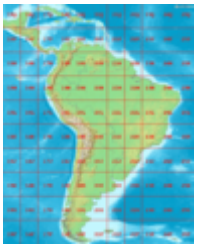
shrunk to cover only open water. Also, in the region around Svalbard, the four grid zones 31X (9° of longitude in width), 33X (12° of longitude in width), 35X (12° of longitude in width), and 37X (9° of longitude in width) are extended to cover what would otherwise have been covered by the seven grid zones 31X to 37X. The three grid zones 32X, 34X and 36X are not used.

- **Grid zones in various parts of the world**

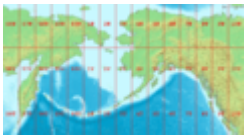
Europe



Africa



South America



Bering Sea with Alaska

Locating a position using UTM coordinates[edit]

A position on the Earth is given by the UTM zone number and the easting and northing planar coordinate pair in that zone. The point of origin) of each UTM zone is the intersection of the equator and the zone's central meridian. To avoid dealing with negative numbers, the central meridian of each zone is defined to coincide with 500000 meters East. In any zone a point that has an easting of 400000 meters is about 100 km west of the central meridian. For most such points, the true distance would be slightly more than 100 km as measured on the surface of the Earth because of the distortion of the projection. UTM eastings range from about 167000 meters to 833000 meters at the equator.

In the northern hemisphere positions are measured northward from zero at the equator. The maximum "northing" value is about 9300000 meters at latitude 84 degrees North, the north end of the UTM zones. In the southern hemisphere northings decrease southward from the equator, set at 10000000 meters, to about 1100000 meters at 80 degrees South, the south end of the UTM zones. For the southern hemisphere, its northing at the equator is set at 10000000 meters so no point has a negative northing value.

The CN Tower is at 43°38′33.24″N 79°23′13.7″W, which is in UTM zone 17, and the grid position is 630084 m east, 4833438 m north. Two points in Zone 17 have these coordinates, one in the northern hemisphere and one in the south; one of two conventions is used to say which:

1. Append a hemisphere designator to the zone number, "N" or "S", thus "17N 630084 4833438". This supplies the minimum information to define the position uniquely.
2. Supply the grid zone, i.e., the latitude band designator appended to the zone number, thus "17T 630084 4833438". The provision of the latitude band along with northing supplies redundant information (which may, as a consequence, be contradictory if misused).

Because latitude band "S" is in the northern hemisphere, a designation such as "38S" is unclear. The "S" might refer to the latitude band (32°N–40°N) or it might mean "South". It is therefore important to specify which convention is being used, e.g., by spelling out the hemisphere, "North" or "South", or using different symbols, such as − for south and + for north.

Simplified formulae[edit]

These formulae are truncated version of Transverse Mercator: flattening series, which were originally derived by Johann Heinrich Louis Krüger in 1912.[9] They are accurate to around a millimeter within 3,000 km of the central meridian.[10] Concise commentaries for their derivation have also been given.[11][12]

The WGS 84 spatial reference system describes Earth as an oblate spheroid along north-south axis with an equatorial radius of $a = 6378.137$ km and an inverse flattening of $1/f = 298.257223563$. Let's take a point of latitude φ and of longitude λ and compute its UTM coordinates as well as point scale factor k and meridian convergence γ using a reference meridian of longitude . By convention,

in the northern hemisphere $N_0 = 0$km and in the southern hemisphere $N_0 = 10000$

km. By convention also $k_0 = 0.9996$and $E_0 = 500$ km.

In the following formulas, the distances are in kilometers. In advance, let's compute some preliminary values:

$$n = \frac{f}{2-f}, A = \frac{a}{1+n}\left(1 + \frac{n^2}{4} + \frac{n^4}{64} + \cdots\right),$$

$$\alpha_1 = \frac{1}{2}n - \frac{2}{3}n^2 + \frac{5}{16}n^3, \alpha_2 = \frac{13}{48}n^2 - \frac{3}{5}n^3, \alpha_3 = \frac{61}{240}n^3,$$

$$\beta_1 = \frac{1}{2}n - \frac{2}{3}n^2 + \frac{37}{96}n^3, \beta_2 = \frac{1}{48}n^2 + \frac{1}{15}n^3, \beta_3 = \frac{17}{480}n^3,$$

$\delta_1 = 2n - \frac{2}{3}n^2 - 2n^3, \delta_2 = \frac{7}{3}n^2 - \frac{8}{5}n^3, \delta_3 = \frac{56}{15}n^3.$

From latitude, longitude (φ,λ) to UTM coordinates (E, N)[edit]

First let's compute some intermediate values:

$t = \sinh\left(\tanh^{-1}\sin\varphi - \frac{2\sqrt{n}}{1+n}tanh^{-1}(\frac{2\sqrt{n}}{1+n}\sin\varphi)\right),$

$\xi' = \tan^{-1}\left(\frac{t}{\cos(\lambda-\lambda_0)}\right), \eta' = \tanh^{-1}\left(\frac{\sin(\lambda-\lambda_0)}{\sqrt{1+t^2}}\right),$

$\sigma = 1 + \sum_{j=1}^{3} 2j\alpha_j \cos(2j\xi')\cosh(2j\eta'), \tau = \sum_{j=1}^{3} 2j\alpha_j \sin(2j\xi')\sinh(2j\eta')$

The final formulae are:

$E = E_0 + k_0 A\left(\eta' + \sum_{j=1}^{3}\alpha_j \cos(2j\xi')\sinh(2j\eta')\right)$

$N = N_0 + k_0 A\left(\xi' + \sum_{j=1}^{3}\alpha_j \sin(2j\xi')\cosh(2j\eta')\right)$

$k = \frac{k_0 A}{a}\sqrt{\{1 + \left(\frac{1-n}{1+n}\tan\varphi\right)^2\}\frac{\sigma^2+\tau^2}{t^2+\cos^2(\lambda-\lambda_0)}},$

$\gamma = \tan^{-1}\left(\frac{\tau\sqrt{1+t^2} + \sigma t\tan(\lambda-\lambda_0)}{\sigma\sqrt{1+t^2} - \tau t\tan(\lambda-\lambda_0)}\right)$

From UTM coordinates (E, N, Zone, Hemi) to latitude, longitude (φ, λ)[edit]

Note: Hemi=+1 for Northern, Hemi=-1 for Southern

First let's compute some intermediate values:

$\xi = \frac{N-N_0}{k_0 A}, \eta = \frac{E-E_0}{k_0 A},$

$\xi' = \xi - \sum_{j=1}^{3}\beta_j \sin(2j\xi)\cosh(2j\eta), \eta' = \eta - \sum_{j=1}^{3}\beta_j \cos(2j\xi)\sinh(2j\eta),$

$\sigma' = 1 - \sum_{j=1}^{3} 2j\beta_j \cos(2j\xi)\cosh(2j\eta), \tau' = \sum_{j=1}^{3} 2j\beta_j \sin(2j\xi)\sinh(2j\eta),$

$\chi = \sin^{-1}\left(\frac{\sin\xi'}{\cosh\eta'}\right).$

The final formulae are:

$\varphi = \chi + \sum_{j=1}^{3}\sigma_j \sin(2j\chi),$

$\lambda_0 = Zone * 6^\circ - 183^\circ$

$\lambda = \lambda_0 + \tan^{-1}\left(\frac{\sinh\eta'}{\cos\xi'}\right)$

$k = \frac{k_0 A}{a}\sqrt{\{1 + (\frac{1-n}{1+n}\tan\varphi)^2\}\frac{\cos^2\xi'+\sinh^2\eta'}{\sigma'^2+\tau'^2}},$

$\gamma = Hemi * \tan^{-1}\left(\frac{\tau' + \sigma'\tan\xi'\tanh\eta'}{\sigma' - \tau'\tan\xi'\tanh\eta'}\right)$