QTL Analysis Pipeline - Use Cases Guide

Quick Start Examples

🚀 Basic eQTL Analysis

Use Case: Standard expression QTL analysis with VCF genotypes

```
bash
```

- `# 1. Edit main config with your file paths`
- `nano config/config.yaml`
-
- `# 2. Update these paths in config.yaml:`
- `input_files:`
- `  genotypes: "your_data/genotypes.vcf.gz"`
- `  covariates: "your_data/covariates.txt"`
- `  annotations: "your_data/annotations.bed"`
- `  expression: "your_data/expression.txt"`
-
- `# 3. Run eQTL analysis`

```
python run_QTLPipeline.py --config config/config.yaml --analysis-types eqtl
```

Expected Output: `results/qtl_results/eqtl_significant.txt` with significant eQTL associations

📋 Use Case 1: GATK4.5 Joint Calling VCF

Scenario

You have genotype data from GATK4.5 joint calling in VCF.GZ format with sample names like `sample_001_L001`.

Configuration

```
bash
```

- `# Use the GATK-optimized configuration`
- `cp config/gatk_joint_calling.yaml config/my_gatk_analysis.yaml`
-
- `# Edit with your paths`

```
nano config/my_gatk_analysis.yaml
```

Key Settings for GATK:

```
yaml
```

- `genotype_processing:`
- `  normalize_chromosomes: true`
- `  chromosome_prefix: "chr"         # Converts "1" → "chr1"`
- `  handle_multiallelic: true       # Splits multi-allelic sites`
- `  remove_phasing: true           # Removes phasing information`

```
  min_call_rate: 0.98           # GATK has high call rates
```

Run Command

```
bash

python run_QTLPipeline.py --config config/my_gatk_analysis.yaml
```

Expected Processing

- ✅ Automatic format detection
- ✅ Chromosome naming normalization
- ✅ Multi-allelic site splitting
- ✅ Phasing removal
- ✅ Quality filtering

📋 Use Case 2: PLINK BED Format

Scenario

You have genotype data in PLINK BED format (.bed, .bim, .fam files).

Configuration

```
bash
```

- `cp config/plink_format.yaml config/my_plink_analysis.yaml`

```
nano config/my_plink_analysis.yaml
```

Key Settings for PLINK:

```
yaml
```

- `input_files:`
- `  genotypes: "your_data/plink_data.bed"  # .bim and .fam must exist`
-
- `genotype_processing:`
- `  auto_detect_format: true      # Automatically converts to VCF`

```
  chromosome_prefix: "none"     # PLINK uses "1" not "chr1"
```

Run Command

```
bash

python run_QTLPipeline.py --config config/my_plink_analysis.yaml
```

Expected Processing

- ✅ Automatic BED → VCF conversion
- ✅ Sample information from .fam file
- ✅ Variant information from .bim file

- ✅ Standardized output format

📋 Use Case 3: Multi-omics Analysis

Scenario

You want to run eQTL, pQTL, and sQTL analyses together to compare molecular QTLs.

Configuration

```
bash
```

- `cp config/multi_omics.yaml config/my_multiomics.yaml`

```
nano config/my_multiomics.yaml
```

Key Settings for Multi-omics:

```
yaml
```

```
input_files:
  expression: "data/expression.txt"
  protein: "data/protein.txt"
  splicing: "data/splicing.txt"

analysis:
  qtl_types: "all"              # Runs all three analyses

plotting:
  plot_types:
    - "manhattan"
    - "qq"
```

```
    - "summary"                 # Cross-analysis comparison plot
```

Run Command

```
bash

python run_QTLPipeline.py --config config/my_multiomics.yaml
```

Expected Outputs

- `eqtl_significant.txt` - Expression QTLs
- `pqtl_significant.txt` - Protein QTLs
- `sqtl_significant.txt` - Splicing QTLs
- `analysis_summary.png` - Comparison of all three

📋 Use Case 4: GWAS + QTL Integration

Scenario

You want to run both QTL analysis and GWAS on the same genotype data.

Configuration

```
bash
```

- `cp config/multi_omics.yaml config/my_gwas_integration.yaml`

```
nano config/my_gwas_integration.yaml
```

Key Settings for GWAS:

```
yaml
```

- `analysis:`
- `  qtl_types: "eqtl"             # Run eQTL analysis`
- `  run_gwas: true                # Enable GWAS`
- `  gwas_phenotype: "data/gwas_phenotypes.txt"`
-
- `gwas:`
- `  method: "linear"              # For continuous traits`

- `# method: "logistic"          # For case-control traits`

```
covariates: true             # Adjust for covariates
```

GWAS Phenotype File Format:

```
text
```

- `sample_id    height    weight    disease_status`
- `sample1      175.2     68.5      0`
- `sample2      168.7     72.1      1`

```
sample3      182.3     75.8      0
```

Run Command

```bash
python run_QTLPipeline.py --config config/my_gwas_integration.yaml
```

Expected Outputs

- QTL results in `qtl_results/`
- GWAS results in `gwas_results/`
- Combined plots showing both analyses

📋 Use Case 5: Large-Scale Study

Scenario

You have thousands of samples and need to optimize for computational efficiency.

Configuration

```
bash
```

- `cp config/large_scale_eqtl.yaml config/my_large_study.yaml`

```
nano config/my_large_study.yaml
```

Optimization Settings:

```
yaml
```

- `qtl:`
- `  permutations: 1000          # Reduced for efficiency (default: 1000)`
- `  # permutations: 100        # For very large studies`
-
- `output:`
- `  remove_intermediate: true     # Save disk space`
- `  compression: true           # Compress large files`
-
- `genotype_processing:`
- `  filter_variants: true`
- `  min_maf: 0.01              # Filter rare variants to reduce load`

```
  min_call_rate: 0.95
```

Run Command

```
bash
```

- `# Test with subset first`
- `python run_QTLPipeline.py --config config/my_large_study.yaml --analysis-types eqtl`
-
- `# Full run`

```
python run_QTLPipeline.py --config config/my_large_study.yaml
```

Performance Tips

- Start with fewer permutations for testing
- Use `--validate-only` first to check inputs

- Monitor memory usage in large runs
- Consider running on HPC cluster for very large datasets

📋 Use Case 6: Fine-Mapping Analysis

Scenario

You want high-resolution QTL mapping with strict quality controls.

Configuration

```
bash
```

- `cp config/fine_mapping.yaml config/my_finemapping.yaml`

```
nano config/my_finemapping.yaml
```

Fine-Mapping Settings:

```
yaml
```

- `qtl:`
- `  cis_window: 500000          # Smaller window for fine-mapping`
- `  permutations: 10000         # More permutations for accuracy`
- `  fdr_threshold: 0.01         # Stricter significance threshold`
-
- `genotype_processing:`
- `  min_maf: 0.01              # Include rare variants`
- `  min_call_rate: 0.99        # High quality requirement`

```
  quality_threshold: 50       # Strict quality filter
```

Run Command

```
bash
```

```
python run_QTLPipeline.py --config config/my_finemapping.yaml
```

Expected Results

- Higher precision QTL mapping
- More accurate FDR estimates
- Better variant-gene pairing

📋 Use Case 7: Tissue-Specific Analysis

Scenario

You have data from specific tissues and want tissue-specific QTLs.

Configuration

```
yaml
```

- # Start with main config and modify:
- qtl:
-   cis_window: 500000   # Smaller window for tissue-specific
-
- genotype_processing:
-   filter_variants: true
-   min_maf: 0.05   # Higher MAF for power in smaller samples
-
- analysis:

```
  qtl_types: "eqtl"           # Focus on expression
```

Run Command

```
bash
```

```
python run_QTLPipeline.py --config config/config.yaml --analysis-types eqtl
```

🛠️ Troubleshooting Common Scenarios

Sample Name Mismatch

Problem: Samples in genotype file don't match phenotype file
Solution:

```
yaml
```

- `genotype_processing:`
- `  match_samples: true`

```
  sample_matching_method: "prefix"  # or "pattern"
```

Memory Issues

Problem: Pipeline runs out of memory
Solution:

```
yaml
```

- `qtl:`
- `  permutations: 100           # Reduce permutations`
- `output:`

```
  remove_intermediate: true   # Clean up files
```

Long Runtime

Problem: Analysis taking too long
Solution:

```
yaml
```

- qtl:
-   permutations: 1000   # Reduce from default
-   cis_window: 500000   # Smaller window
- genotype_processing:

```
min_maf: 0.05              # Filter more variants
```

🎯 Quick Reference Commands

Use Case	Command
Basic eQTL	`python run_QTLPipeline.py --config config.yaml --analysis-types eqtl`
GATK VCF	`python run_QTLPipeline.py --config gatk_joint_calling.yaml`
PLINK BED	`python run_QTLPipeline.py --config plink_format.yaml`
Multi-omics	`python run_QTLPipeline.py --config multi_omics.yaml`

With GWAS	`python run_QTLPipeline.py --config config.yaml --run-gwas`
Validate Only	`python run_QTLPipeline.py --config config.yaml --validate-only`
Large Study	`python run_QTLPipeline.py --config large_scale_eqtl.yaml`

-