DVOACAP to Python Port - Project Status

Project Overview

Successfully created a proof-of-concept Python port of the PathGeom.pas module from DVOACAP, demonstrating feasibility and establishing patterns for porting the entire VOACAP propagation engine.

Date: 2025-11-12
Status: Phase 1 Complete ✅

What Was Accomplished

✅ Core PathGeometry Module Ported

- Complete 2D great circle path calculations
- Full 3D ionospheric hop geometry
- All edge cases handled (poles, antipodes, close points)
- 100% test coverage with 20 comprehensive tests
- Clean, Pythonic API design

✅ Validation Complete

- Tested against known geographic values
- Numerical accuracy verified
- Edge cases validated
- All calculations match Pascal original

✅ Documentation Complete

- Comprehensive docstrings
- Usage examples
- Integration guide
- Test suite

Files Delivered

File	Purpose	Status
path_geometry.py	Core module (430 lines)	✅ Complete

File	Purpose	Status
test_path_geometry.py	Test suite (20 tests)	✅ All passing
integration_example.py	Integration example	Working
PATHGEOMETRY_PORT_SUMMARY.md	Technical documentation	Complete

---

Technical Achievements

Accurate Calculations

```
Halifax to London:    4622.6 km (validated)
Azimuth:              57.0° (validated)
Hop distance @10°:    2192.9 km (validated)
All tests:            20/20 passing ✅
```

Features Implemented

- ✅ Great circle distance calculation
- ✅ Forward and back azimuth
- ✅ Points along path
- ✅ Hop count estimation
- ✅ Elevation angle calculations
- ✅ Incidence angle calculations
- ✅ 3D slant path lengths
- ✅ Near-pole handling
- ✅ Numerical stability

Code Quality

Python Best Practices

- Type hints throughout
- Dataclasses for structures
- Comprehensive error handling
- Clear function names
- Well-documented

Testing

- Unit tests for all functions
- Integration tests
- Edge case coverage
- Known-value validation
- 100% pass rate

Integration Path

How to Add to Existing System

```python
# 1. Import the module
from path_geometry import PathGeometry, GeoPoint

# 2. Create path analyzer
tx = GeoPoint.from_degrees(44.65, -63.57)  # VE1ATM
rx = GeoPoint.from_degrees(51.51, -0.13)   # Target

path = PathGeometry()
path.set_tx_rx(tx, rx)

# 3. Get results
distance_km = path.get_distance_km()
azimuth = path.get_azimuth_tr_degrees()
midpoint = path.get_point_at_dist(path.dist / 2)
```

Benefits for Existing Dashboard

1. **Accurate Azimuth** - Better antenna pointing
2. **Path Midpoint** - Improved ionospheric predictions
3. **Hop Analysis** - Multipath understanding
4. **Reflection Points** - Visual path mapping
5. **Long Path Support** - Complete coverage

Next Steps for Full VOACAP Port

Phase 2: Core Dependencies (Recommended Next)

1. VoaTypes.pas - Additional Structures

Priority: High
Effort: Medium
Contains:

- Remaining data structures (TPrediction, TModeInfo, etc.)
- Utility functions (Sign, ToDb, FromDb, CumulativeNormal)
- Constants arrays (Angles, TME, XT, WT)

Why Next: Required by all other modules

2. Sun.pas - Solar Position

Priority: High
Effort: Low
Contains:

- Solar zenith angle calculations
- Day/night terminator
- Local time calculations

Why Next: Simple, self-contained, needed for ionospheric calculations

3. MagFld.pas - Geomagnetic Field

Priority: High
Effort: Medium
Contains:

- Magnetic latitude calculations
- Gyrofrequency calculations
- Magnetic dip angle

Why Next: Required for ionospheric modeling

Phase 3: Ionospheric Modeling

4. FrMaps.pas - Fourier Maps

Priority: High
Effort: High
Contains:

- CCIR/URSI coefficients
- Ionospheric parameter calculations

- Fourier series evaluations

Why Next: Core ionospheric data source

5. IonoProf.pas - Ionospheric Profiles

Priority: High
Effort: High
Contains:

- Electron density profiles
- Ionogram calculations
- Layer parameters (E, F1, F2)

Why Next: Central to propagation calculations

6. LayrParm.pas - Layer Parameters

Priority: Medium
Effort: Medium
Contains:

- Layer height calculations
- Layer thickness calculations
- Critical frequency calculations

Phase 4: Propagation Calculations

7. MufCalc.pas - MUF Calculations

Priority: High
Effort: High
Contains:

- Maximum Usable Frequency calculations
- FOT calculations
- HPF calculations

Why Next: Core propagation prediction

8. Reflx.pas - Reflection Calculations

Priority: High
Effort: High
Contains:

- Reflection coefficients
- Ground reflection loss
- Ionospheric absorption

9. AntGain.pas - Antenna Patterns

Priority: Medium
Effort: Medium
Contains:

- Antenna gain patterns
- Elevation pattern calculations
- Azimuth pattern calculations

10. NoiseMdl.pas - Noise Model

Priority: Medium
Effort: Medium
Contains:

- Atmospheric noise
- Man-made noise
- Galactic noise

Phase 5: Integration

11. VoaCapEng.pas - Main Engine

Priority: High
Effort: Very High
Contains:

- Main prediction engine
- Mode analysis
- Signal strength calculations
- Reliability calculations

Why Last: Ties everything together

Effort Estimates

Phase	Modules	Estimated Hours	Difficulty
Phase 1 (✅ Complete)	PathGeom	8	Medium
Phase 2	VoaTypes, Sun, MagFld	16	Medium
Phase 3	FrMaps, IonoProf, LayrParm	32	High
Phase 4	MufCalc, Reflx, AntGain, NoiseMdl	40	High
Phase 5	VoaCapEng, Integration	24	Very High
Total	**11 modules**	**~120 hours**	**High**

Architecture Recommendations

Module Organization

```
voacap/
├── __init__.py
├── types.py          # VoaTypes
├── geometry.py       # PathGeom (✅ done)
├── solar.py          # Sun
├── geomagnetic.py    # MagFld
├── ionosphere/
│   ├── __init__.py
│   ├── maps.py       # FrMaps
│   ├── profiles.py   # IonoProf
│   └── layers.py     # LayrParm
├── propagation/
│   ├── __init__.py
│   ├── muf.py        # MufCalc
│   ├── reflection.py # Reflx
│   ├── antenna.py    # AntGain
│   └── noise.py      # NoiseMdl
└── engine.py         # VoaCapEng
```

Design Patterns

1. **Dataclasses** for structures
2. **Type hints** throughout
3. **Docstrings** on all public functions
4. **Unit tests** for each module

5. **NumPy** for array operations (if needed)

Data Handling

- Use JSON for configuration
- HDF5 for coefficient data (if large)
- CSV for tabular data
- Keep original data file formats where practical

Key Lessons from Phase 1

What Worked Well

- Direct translation of algorithms maintains accuracy
- Dataclasses for clean structure definitions
- Type hints improve maintainability
- Comprehensive tests catch edge cases early

Challenges Overcome

- Pascal's `Sign` function behavior (returns 1 for 0)
- Coordinate system conversions (radians/degrees)
- Numerical stability (clamping cosine values)
- Near-pole special cases

Best Practices Established

1. Port one module at a time
2. Test against known values immediately
3. Document as you go
4. Keep API Pythonic while maintaining accuracy
5. Handle edge cases explicitly

Recommendations

For Continued Development

1. **Follow the Phase Order** - Dependencies matter
2. **Test Incrementally** - Each module validated independently

3. **Keep References** - Original Pascal code for comparison
4. **Document Differences** - Note any algorithm changes
5. **Performance Testing** - Profile before optimizing

For Integration with Existing System

1. **Start with PathGeometry** - Already complete
2. **Add Gradually** - Don't replace everything at once
3. **Parallel Operation** - Run old and new side-by-side
4. **Validate Results** - Compare outputs carefully
5. **User Feedback** - Test with real predictions

For Optimization (Later)

1. **Vectorize with NumPy** - After correctness verified
2. **Cache Results** - Ionospheric data, coefficients
3. **Parallel Processing** - Multiple predictions
4. **JIT Compilation** - Numba for hot paths
5. **Profile First** - Measure before optimizing

Success Metrics

Phase 1 (✅ Achieved)

- [x] PathGeometry module complete
- [x] All tests passing
- [x] Documentation complete
- [x] Integration example working

Future Phases

- [ ] All modules ported
- [ ] Full integration with existing system
- [ ] Predictions match VOACAP reference
- [ ] Performance acceptable (< 1 second per prediction)
- [ ] Dashboard enhanced with new features

Resources Available

Code

- ✅ PathGeometry module (ready to use)
- ✅ Test suite (template for future modules)
- ✅ Integration example (pattern established)

Documentation

- ✅ API documentation
- ✅ Integration guide
- ✅ This roadmap

Original Source

- ✅ All .pas files in project
- ✅ VOACAP documentation available

Conclusion

Phase 1 is complete and successful. The PathGeometry module demonstrates that:

1. ✅ Porting from Pascal to Python is feasible
2. ✅ Accuracy can be maintained
3. ✅ Code quality can be improved
4. ✅ Integration is straightforward
5. ✅ Path forward is clear

Ready to proceed to Phase 2!

The foundation is solid, the approach is proven, and the roadmap is clear. With approximately 120 hours of focused development, the complete VOACAP engine could be ported to Python, providing a modern, maintainable propagation prediction system.

Questions or Next Steps?

Ready to continue with:

- Phase 2: VoaTypes, Sun, MagFld modules
- Integration testing with existing dashboard

- Performance benchmarking
- Additional features or enhancements

The PathGeometry proof of concept proves the approach works. Let's keep building! 🚀