Dynalog Analysis Package for Matlab and Octave

Release 0.2

Michael Hughes

IMPORTANT

This software is not associated with, or approved by, Varian or my employer in any way. It is provided 'as-is' and is to be used at your own risk. I assume no responsibility whatsoever for its use, and make no guarantees, expressed or implied, about its quality, reliability, or any other characteristic. The software is made available for educational and research purposes only. It has not passed through a rigorous quality control system and its use is not recommended for any purposes which could affect patient care or treatment.

LICENSE

This software can be redistributed and/or modified freely provided that any derivative works bear some notice that they are derived from it, and any modified versions bear some notice that they have been modified. I would appreciate acknowledgement if the software is used in any published work.

Contents

Introduction

The Dynalog Analysis Package is a set of functions for Octave/Matlab. These functions been written for Octave v3.2.4, but should run under Matlab with minimal modification. They can be used to interrogate dynalog files generated by the Varian Clinac 4D Treatment System. I wrote the functions so that I could analyse large numbers of dynalog files as part of a project looking into their use in quality assurance. I hope the software will of use to other physicists or engineers working in the field of IMRT. Please note that, since there is no GUI, the user will require a working knowledge of matlab/octave and a general understanding of dynalog files.

I am happy to discuss the software or the project it was developed for (time permitting) and can be contacted at michael.robert.hughes@gmail.com. If people have substantial additions to the software I would be happy to include them in the 'official' package. If you spot any bugs (there are undoubtedly many), feel free to let me know and I will hopefully get round to fixing it. Or better still, fix it yourself and send me a copy of your code!

Version History

Changes from release 0.1:

1. dynRead optimised.
2. dynFluence now allows for interpolation (no longer compatible with scripts written for previous version)
3. dynGammaQuick added.
4. When calling dynLeafSpeed, can now specify total treatment time.
5. dynReadMLC added.
6. dynDoseRate added.
7. Examples expanded to demonstrate more functionality.

Structure of Dynalog Files

The following structure of .dlg files is assumed. If the file structure on your system varies, you will need to edit dynRead.m.

Line 1: Version Letter
Line 2: Patient Name/ID
Line 3: Various version numbers
Line 4: Plan Tolerance
Line 5: Number of Leaves
Line 6: Clinac scale
Lines 7+: Each line corresponds to one dose fraction, sampled every 50 ms.
- Column 1: Dose fraction, ranges from 0 to 25000 (i.e. 25000 is 100% of MU)
- Column 2: DVA Segment
- Column 3: Beam Hold-off flag (1=hold off)
- Column 4: Beam On flag (1=beam on)
- Column 5: Dose fraction of previous segment
- Column 6: Dose fraction of next segment
- Column 7: Gantry Rotation (10th of a degree)
- Column 8: Collimator Rotation (10th of a degree)
- Column 9: Y1 Jaw Position (10^{th} of a mm)
- Column 10: Y2 Jaw Position (10^{th} of a mm)
- Column 11: X1 Jaw Position (10^{th} of a mm)
- Column 12: X2 Jaw Position (10^{th} of a mm)
- Column 13: Carriage planned position (100^{th} of a mm)
- Column 14: Carriage actual position (100^{th} of a mm)
- Column 15: Planned position for leaf 1
- Column 16: Actual position for leaf 1
- Column 17: Previous field position for leaf 1
- Column 18: Next field position for leaf 1
- Columns 19+ Repeat columns 13-17 for other leaves.

Note that units are converted to mm or degrees by **dynRead**.

Potential Modifications

Several physical parameters are currently hard-coded and will need editing if your system differs. The key parameters can be set as global constants using **dynConstants.** Edit this file as required and then call it at the beginning of your scripts.

A (non-exhaustive) list of the possible changes required is below:

1. Structure of Dynalog files. If your files are not structured as described above you will need to edit **dynRead**.

2. Projected leaf size at isocenter. Set as 1.96614 x real leaf size. Set in **dynConstants**. Affects data read-in by dynRead and all subsequent calculations performed on data.
3. Number and size of leaves. Set in **dynConstants**. Only affects **dynFluence**.
4. Time between dose fraction records set as 50 ms. Set in **dynConstants**. Only affects **dynLeafSpeed**.

Error Checking

There is currently no error checking on any functions; invalid parameters may result in run-time errors or inaccuracies in results.

Example Programs

Two programs are provided which demonstrate how to use to use the various functions. See the header comments in the .m files for more details.

exampleSummary: Produces a summary of data from all dynalog files in the current directory, including data such as RMS leaf error. Also produces plots of fluence and performs a gamma analysis.

exampleSpeeds: Produces plots showing speeds of leaves and a leaf speed histogram.

Functions Summary

dynConstants:	Allows some global constants to be set.
dynDoseRate:	Calculates dose rate at all fractions if total MUs are known.
dynError:	Calculates matrix of all leaf position errors at all dose fractions.
dynFluence:	Generates the fluence maps expected from the actual and planned leaf positions and dose rates.
dynGamma:	Generates gamma index matrix for two fluence maps.
dynGammaQuick:	Provides a rapid estimation of gamma index matrix.
dynGap:	Returns gap between two banks of leaves for all leaves and all dose fractions.
dynGapAbs:	Returns magnitude of gap between two banks of leaves for all leaves and all dose fractions.
dynGapCheck:	Checks gap errors against a specified tolerance.
dynGapError:	Calculates leaf gap error for all leaves and at all dose fractions.
dynIsBeamOn:	Returns true if beam was on for specified dose fraction.
dynIsLeafMoving:	Returns true if specified leaf moved during field.
dynLeafCheck:	Checks leaf errors against a specified tolerance.
dynLeafSpeed:	Calculates matrix of all leaf speeds at all dose fractions.
dynNumBeamOn:	Returns number of fractions with beam on.

Function Descriptions

dynConstants

Description: Sets some constants, including leaf widths and leaf sizes. Edit this file as required and then call it at the start of your scripts to model different systems.

Syntax: dynConstants

dynDoseRate

Description: Calculates dose rate at all dose fractions. Returns a vector with length one less than number of fractions. Dose rates are expressed as (fraction of total MU) per minute.

Syntax: dynDoseRate = dynDoseRate(dynData)

dynData: struct from dynRead;

dynError

Description: Returns matrix of differences between actual and planned positions.

Syntax: error = dynError(dynData)

dynData: struct from dynRead;

dynFluence

Description: Generates a fluence map from two dynData structs. Fluence values

are scaled essentially arbitrarily and cannot be directly compared with other fluence maps. Setting a large timeInterp improves accuracy but slows calculation.

Syntax: [mapPlanned mapActual] = dynFluence(dynDataA, dynDataB, scaleFactor, timeInterp);

mapPlanned :	Fluence map which was planned
mapActual :	Fluence map which was generated
dynDataA/B :	Structs from dynRead for banks A and B
scaleFactor:	Pixels per mm in fluence map
timeInterp:	Interpolates by this factor between time points.

dynGamma

Description: Calculates gamma analysis map between two fluence maps. For description of technique see: Low, D.A., et al., Med. Phys. 25, p656 (1998).
IMPORTANT: dynGamma will tend to over-estimate the gamma index when the matrix pitch is comparable to the distance criterion. To avoid this, it is necessary to increase the interpFactor argument. However, this also dramatically increases the time required to compute the gamma analysis. Also note that the matrix is interpolated by a factor of 2^interpFactor.

Syntax: gammaMap = dynGamma(mapA, mapB, doseCriterion, distanceCriterion, normDose);

gammaMap:	2D Array containing gamma indices
mapA, mapB:	2D Arrays containing fluence maps
doseCriterion:	The dose criterion to test (as a fraction of normDose)
distanceCriterion:	The distance criterion to test (in pixels)
normDose:	The fluence value which will be taken as 100%
interpFactor:	Matrices are interpolated by 2^interpFactor

dynGammaQuick

Description: Produces an estimate of the gamma analysis map between two fluence maps by examining the local fluence gradient. In is only an approximation, but the function offers at least an order of magnitude speed increase over dynGamma. A full description of this technique, including a description of its limitations, is given in Bakai, A. et al., Phys. Med. Biol. 48 p3543 (2003). In particular it should be noted that the approximation becomes less accurate as the second

derivative of the fluence becomes larger, and tends to the true value as the second derivative tends to zero.

Syntax: gammaMap = dynGamma(mapA, mapB, doseCriterion, distanceCriterion, normDose);

gammaMap: 2D Array containing gamma indices
mapA, mapB: 2D Arrays containing fluence maps
doseCriterion: The dose criterion to test (as a fraction of normDose)
distanceCriterion: The distance criterion to test (in pixels)
normDose: The fluence value which will be taken as 100%

dynGap

Description: Calculates the gap between leaves of two banks at all dose fractions. Requires that Bank A positions < Bank B positions, otherwise returned gaps will be negative. If relationship between banks is unknown, use dynGapAbs which calculates the magnitude of the gap only.

Syntax: gap = dynGap(dynDataA, dynDataB);

dynDataA/B : Structs from dynRead for Banks A and B

Returns: Struct of two arrays:
.planGap: planned gap
.actualGap: actual gap

dynGapAbs

Description: Calculates magnitude of gap between leaves of two banks at all dose fractions. To determine sign of gap (which should be +ve unless there is an error) use dynGap.

Syntax: gapAbs = dynGapAbs(dynDataA, dynDataB);

dynDataA/B: Structs from dynRead for Banks A and B

Returns: Struct of two arrays:
.planGap: planned gap
.actualGap: actual gap

dynGapCheck

Description: For each leaf, checks gap size error at each dose fraction against a tolerance. Returns a vector, true for leaves which have fraction 'fractions' of dose fractions with an error greater than or equal to 'tolerance'.

Syntax: gapCheck = dynGapCheck(dynDataA, dynDataB, tolerance, segments);

gapCheck:	vector of booleans (1 to .numLeaves)
dynDataA/B:	structs from dynRead for Banks A and B
tolerance:	error tolerance
fractions:	fraction of dose fractions which are allowed to be out of tolerance for each leaf.

dynGapError

Description: Returns a matrix of differences between planned and actual gap sizes.

Syntax: gapError = dynGapError(dynDataA,dynDataB);

dynDataA/B:	structs from dynRead for Banks A and B

dynIsBeamOn

Description: Checks if beam was on for specified dose fraction. Returns 1 if Beam On, 0 if Beam Off

Syntax: isBeamOn = dynIsBeamOn(dynData, fraction);

dynData:	struct from dynRead
fraction:	dose fraction to check

dynIsLeafMoving

Description: Checks if leaf was planned to move at any point during field. Returns 1 if moving, 0 if not moving.

Syntax: isMoving = dynIsLeafMoving(dynData, leaf);

dynData:	struct from dynRead
leaf:	leaf number to check (refers to position in array, not .leafNumber field). Can be a vector.

dynLeafCheck

Description: Checks each leaf against a tolerance. Returns positive if fractions of dose fractions show a leaf position error greater than tolerance.

Syntax: leafTest = dynLeafCheck(dynData, tolerance, segments);

leafTest:	vector giving with boolean value for each leaf
dynData:	struct from dynRead
tolerance:	error tolerance
fractions:	fraction of dose fractions which are allowed to be out of tolerance

dynLeafSpeed

Description: Calculates speed of each leaf during each dose fraction. If using with readMLC then totalTime should be specified.

Syntax: leafSpeed = dynLeafSpeed(dynDataA, [totalTime]);

leafSpeed:	2D Array of size (dynData.numFractions - 1) x (dynData.numLeaves) containing leaf speeds in mm/s for all fractions.
dynData:	struct from dynRead.
totalTime:	total treatment time. If left blank, the interval between each fraction is assumed to be 50ms (or the value set in dynConstants).

dynNumBeamOn

Description: Returns the number of Beam-Ons which occurred.

Syntax: numHoldOff = dynNumBeamOn(dynData);

dynData:	struct from dynRead

dynNumHoldOff

Description: Returns the number of Beam Hold Offs which occurred.

Syntax: numHoldOff = dynNumHoldOff(dynData);

dynData:	struct from dynRead

dynOnlyBeamOn

Description: Removes fractions in which beam was planned to be off from dynData struct. Does NOT remove beam hold-offs (i.e. unplanned beam offs).

Syntax: dynDataBeamOn = dynOnlyBeamOn(dynData);

dynDataBeamOn:	output dynData struct
dynData:	struct from dynRead

dynOnlyMoving

Description: Removes leaves which do not move during treatment from dynData struct. Use .leafNumber field to determine original leaf number.

Syntax: dynDataMoving = dynOnlyMoving(dynData);

dynDataMoving:	output dynData struct
dynData:	struct from dynread

dynRead

Description: Reads in data from Varian Dyanlog File to a struct array. You will need to execute this function for both bank files.
Thanks to Ted Fisher for optimising this function.

Syntax: dynData = dynRead(fileName);

filename:	name of dynalog file.

Returns: Struct array with elements:

.mlcfile:	name of file data was read from
.version:	version number of file
.planTolerance:	plan tolerance quoted in file
.numLeaves:	number of leaves in plan

.numFractions:	number of dose fractions (essentially field treatment time / 50 us).
.planPosition:	planned position of leaf
.actualPosition:	position leaf was actually at
.y1JawPosition:	y1 jaw position in mm
.y2JawPosition:	y2 jaw position in mm
.x1JawPosition:	x1 jaw position in mm
.x2JawPosition:	x2 jaw position in mm
.carriagePlanPosition:	carriage planned position in mm
.carriageActualPosition:	carriage actual position in mm
.gantryRotation:	gantry rotation in degrees
.collimatorRotation:	collimator rotation in degrees

dynReadMLC

Description: Reads in data from Varian MLC File to a struct array. The struct array is compatible with that produced by dynRead and it can be used as an input to all functions that accept a dynData struct (although this will not make physical sense in some cases. You will need to execute this function for both banks. When using this function, 'dose fractions' should be taken to read 'segments'.
IMPORTANT: This function is currently very slow.

Syntax: dynData = dynReadMLC(filename, bank);

filename:	name of dynalog file.
bank:	'A' or 'B'

Returns: Struct as for dynRead. However only planPosition, planTolerance, numLeaves and numFractions (i.e. number of segments) are populated. actualPosition is set as a copy of planPosition.

dynRMSGapError

Description: Calculates RMS of difference between actual and planned gap sizes for all leaves and dose fractions.

Syntax: RMSGapError = dynRMSGapError(dynDataA, dynDataB);

dynDataA/B:	struct from dynData for banks A and B.

dynRMSError

Description: Calculates RMS of difference between actual and planned positions for all leaves and dose fractions.

Syntax: RMSError = dynRMSError(dynDataA, dynDataB);

dynDataA/B: struct from dynData for banks A and B.

References

Please see the following for more information of Dynalog files and their use in IMRT QA:

Aydogan B., Li, J. and Smith, B., *DynaLog File Analysis for IMRT Delivery Verification*, Med Phys **37**, 3228, 2010

Kumar, M.D., Thirumavalavan, N., Krishna , D.V. and Babaiah , M., *QA of intensity-modulated beams using dynamic MLC log files, J Med Phys **31**, 36, 2006*