MICROARRAY TECHNOLOGY

Production, Use, and Analysis

Principles of Computational Biology

Final Project

December 2015

[1]Nabil Azamy, Kevin Brown, Nicholas Shefte

[1] Order of authorship is not an indication of degree of contribution

Introduction

DNA microarray technology is a high-throughput method for obtaining the information necessary to make inferences from genomic expression. Microarray technology encompasses a diverse set of production methods, tools, and analytical techniques which continue to evolve and be iterated upon. The fabrication of a microarray involves the use of a substrate with a biological material, such as single-stranded DNA (ssDNA) probes, adhered to the substrate in distinct and identifiable locations. In the case of DNA microarrays, the test material used in microarray analysis is a solution consisting of ssDNA targets labeled with a detectable marker. The marker used on the target ssDNA denotes the source of the DNA, and is used as the measurable variable to derive experimental information. The target solution is washed over the microarray substrate where complementary target and probe strands will hybridize in a competitive manner under conditions where greater concentrations of one labeled target will more likely hybridize to a probe than labeled targets in lower concentrations. The microarray is analyzed by a device that outputs the intensity of the competing dyes as locations on a matrix. The interpretation of the results allow researchers to make inferences on genetic composition or expression based on the design of the experiment. A common use of microarray technology is in the pursuit of gene expression research (Draghici, 2012; Watson, 2013).

Uses of Microarray Technology

Broadly, the use of DNA microarrays in gene expression research would involve a substrate chip containing probes from genome of a target organism. The target solution may be created from mRNA extracted from cells labeled to indicate different states, such as different source tissues or from the same tissue under different conditions. The mRNA would then be reverse-transcribed and labeled with a detectable marker. The result of the analysis would provide insight in the differences in gene expression from the different sources of mRNA given the assumptions that the quantity of mRNA was directly proportional to the level of gene expression under both states, and that the probes on the substrate are appropriately complementary to the DNA target solutions to assure competitive hybridization (Draghici, 2012; Watson2013).

This kind of investigation into gene expression can be utilized towards a number of solvable problems. DNA microarrays have been used to determine new reference genes for species of interest. An organism's transcriptome can be analyzed across a variety of conditions. Genes that exhibit expression fold-changes of low magnitude, or are ideally constant, can be used as potential reference genes (Li, 2015). This method provides a fast alternative to the previous method, which involved comparing genes using PCR procedures. There is growing evidence that reference genes determined through microarray analysis perform better for the purposes of RT-qPCR normalization (Hruz, 2011).

There are many other common uses of microarrays. Data can be compared between microarrays or within two-channel microarrays to measure differential expression between tissues or cell-states. Samples from unknown tissues may be determined through clustering algorithms, or foreign and contaminating DNA could be identified using similar methods.

Microarray Fabrication

There are multiple technologies that are used to create a microarray substrate. The substrate itself is a solid material such as glass or plastic that has an array of DNA probes adhered to known locations on its surface. Three methods used for creating the DNA array can be categorized as contact, non-contact and *in situ* (Dufva, 2005).

The contact methods of fabricating an array involve the use of a machine-operated pin, with heads 60 to 200 µm in diameter, or a stamp. In the case of pins, the apparatus is dipped into wells containing a solution of known DNA probes, then briefly placed in contact with the surface of the chip. The shape of the pin head, the surface tension of the solution, and velocity of the apparatus allow for the transfer of the selected DNA probes to the substrate. In the case of stamps, an 'ink' mixture of DNA probe material and thiols is brought into contact with the stamp where it is temporarily transported by the stamp's hydrophilic surface (Barbulovic-Nad, 2006). Due to the serial nature of this method, production of microarrays is slow and the spot density of the DNA deposits is limited.

BeadArray fabrication, in use in microarrays produced by Illumina®, uses a substrate consisting of either fiber-optic bundles or silica slides. The surface of the substrate is etched with an array of wells set to accommodate 3 µm beads. The beads are covered with multiple copies of a DNA probe, with each nucleotide strand including a unique address sequence. The beads are spread on to the substrate and settle into the wells. While the position of each probe is not controlled, the address sequence provides the position of the probe when the chip is analyzed. This method of microarray fabrication allows for the production of high-density chips (Draghici, 2012).

Non-contact fabrication involves machines, modeled after inkjet printers, position an 'ink' solution containing DNA probe sequences that is dispensed over a known locations on the substrate (Barbulovic-Nad, 2006). These methods are susceptible to contamination, damage to the probes, and imprecise probe placement.

In situ microarray synthesis is the construction of the DNA probe on the chip itself, rather than creating and cloning probes beforehand. Three technologies that follow the *in situ* method of fabrication involve photolithographic masks, single-nucleotide inkjet printers, and electrochemical substrates. *In situ* methods, generally, are capable of achieving higher array densities than contact or noncontact cDNA methods.

The use of photolithographic masks involve the repetition of using light targeted by a mask onto a specific area where the next nucleotide of the growing probe is desired to be added. The light eliminates the protective group on the tail of the probe, which allows the application of another nucleotide which will then have its own protective group. In practice, this method limits probe length to approximately 25 nucleotides (Dufva, 2005).

Single-nucleotide inkjet printers involve the use of nano-scale machines that deposit a single nucleotide to a known location where it will bind to a growing sequence. This method limits probe lengths to approximately 60 nucleotides (Agilent Technologies, 2015).

The third *in situ* method requires substrates embedded with electrodes. As solutions containing a population of one nucleotide are washed over the substrate, the electrodes are activated at the desired positions to bind the nucleotide to the growing strand. (Draghici, 2005).

Chips manufactured using *in situ* methods use two different kinds of probes per gene. The probes, perfect match and mismatch, are positioned near each other and differ by one nucleotide. Unlike cDNA microarray chips, only one sample is analyzed per array rather than two.

Analysis of Microarrays

The process of scanning a microarray and translating the analog properties of the chip into digital data involves solving the borders surrounding each spot and translating the intensity of the detectable marker at each spot. The exact method of accomplishing this varies between machines and manufacturers.

Finding the edges of individual spots is a challenge generally associated with arrays generated with contact or noncontact methods containing cDNA probes. Microarray scanners are equipped with micron-level resolution image detection. The images are filtered to reduce uneven edge boundaries. A pattern-recognition algorithm then assigns attributes to each spot, such as center coordinates, and creates new borders using a fixed radius from the center (Figueroa, 2008).

Quantification of spot intensity is done by storing color data in each pixel of a spot. The memory of the scanning machine determines the number of distinct levels of color that can be differentiated. The color depth is stored bit-wise, per pixel, where eight bits allows the distinction between 256 colors and 24 bits allows the distinction between 16,777,216 colors (Draghici, 2012).

Typically, intensity is calculated using two channels where proprietary chips produced using *in* situ methods may be calculated using one channel. Two-channel array technology calculates the intensity of each marker at each spot. As both markers are being quantified, the values can be used to infer the quantity of hybridization that has occurred at each spot. Single-channel technology compares the intensities positioned at the perfect-match probes and mismatch-probes and calculates an expressed value using formula (1). Single-channel arrays are only able to output the relative amount of probe hybridization and cannot be used to infer quantities of target DNA sequences at each spot (Affymetrix, 2005; Draghici, 2012).

Formula (1)

$$AvgDiff = \frac{\sum_i^N (PM_i - MM_i)}{N}$$

PM <- Perfect Match probe
MM <- Mismatch probe
N <- Number of probes

Challenges Using Microarrays

The use of microarrays present challenges of data normalization and noise reduction. The choice of probe sequences can yield unexpected results when developing an experimental protocol. As probes are short sequences of the genes they represent, DNA that has been processed as a splice variant mRNA or undergone a point mutation in that region may not be properly selected during the hybridization process, consequently resulting in a false low signal for that particular gene. This can be especially problematic when using transgenic specimens (Tomita, 2015). Other broad sources of variability also affect the measurements such as humidity and light, which influence the stability of dyes commonly used on cDNA probes (Amersham Biosciences, 2003).

The variability between arrays and the environmental factors require background correction within each assay and data normalization across multiple assays. Additionally, determinations must be made about low spot signals and outlier data. This variability can come from sources already mentioned or from biases innate to a specific array or scanner.

The use of multiple microarrays also produces an abundance of data, which can require additional time to analyze and risk influencing results with chips that produce data that isn't actually representative of the sample. In these cases, determinations must be made about which chips to use in the calculations.

As all data derived from the use of microarrays can only be used to infer the experimental reality, results must often be verified through the use of PCR techniques to obtain data that isn't confounded by the sources of variability unique to microarrays.

Current Tools in Microarray Analysis

The R programming language and related software environments provide a wide array of tools and techniques that can be used in the analysis of microarrays. Bioconductor, a suite of tools designed to integrate in R environments, provide greater options for use explicitly in bioinformatics including in the analysis of microarrays.

The Bioconductor package, marray, includes powerful tools for data normalization. Mathematical manipulation of the Cy3 and Cy5 dyes used with cDNA arrays corrects the array output, which is necessary as environmental conditions and machine biases influence dye signals. Typically, red intensities bias lower than green intensities (Yang, 2015). One method of within-array normalization is to compute MvA values with formula (2) over local subsets of the array. This method assumes that the mean should not vary greatly on a large sample set; however, a significant number of high or low intensities reduce the effectiveness of this normalization method.

The scale normalization method calculates the deviation from the mean for each value using formula (3) following the constraint that the overall median is set to 0. Scale normalization can be conducted on local groups or globally on the array. It may also be conducted after other forms of local normalization (Park, 2003).

Data may also be normalized based on the intensity of each spot using LOWESS (LOcally WEighted Scatterplot Smoother) smoothing. LOWESS smoothing, based on linear and nonlinear regression, determines a robust linear fit at each point. The local subsets for each point are determined using the k-nearest neighbors (KNN) algorithm. LOWESS normalization is recognized as being computationally expensive due to requiring the use of the KNN algorithm and then calculating the normalized values for each point in relation to each other. There are multiple approaches to LOWESS smoothing, resulting in different tools of different complexity using a variety of formulae in their calculations (Draghici, 2012).

Formula (2)

$$M = log_2 \frac{R}{G}$$

$$A = log_2 \sqrt{R * G}$$

R <- Red

G <- Green

Formula (3)

Median Absolute Deviations = median{$|x_1$-m|…$|x_n$-m|}

Single-channel, oligonucleotide microarrays present their own normalization challenges. A popular algorithm to use is MAS5 from Affymetrix, although there are more. The MAS5 algorithm normalizes from within an array, rather than between each array, by making a log-transformation of the signals and subtracting the signal of the mismatch (MM) probes from the perfect match (PM) probes. MAS5 uses the formula in formula (4) (Irizarray, 2003). The level of adjustment to the CT variable, when the MM signal is greater than or equal to the PM signal, is dependent on the exact model used in the MAS5 implementation. Even with robust correction, low PM signals are a weakness of the MAS5 algorithm requiring the need to remove probes that confound the algorithm in a statistically sound manner (Pepper, 2007).

Formula (4)

$$log(PM_{ij} - CT_{ij})$$

PM <- Perfect Match

CT <- MM if MM < PM, or adjusted to be < PM if MM ≥ PM

Project Algorithm

This tool uses the Genepix Results (gpr) data file produced by a GenePix two-channel microarray scanner. It would be utilized after the data normalization step where it would calculate summary statistics from the file and output them in Excel (xls) format spreadsheets.

This tool is useful for the pre-filtering of data and assists in the determination on selecting or rejecting subsets of the array signals. This can be accomplished using the statistics from this tool to determine what to use as input for a Pearson correlation analysis. In addition, the statistics can be directly used as input to make determinations about outliers, such as with the use of Mean vs Coefficient of Variation plot.

In the case of analyzing patient samples in the pursuit of diagnosing known diseases, this tool could assist in filtering differentially expressed genes known to be representative of different diseases which would allow for less ‘noisy’ sample data to be used in further analysis.

Algorithm Analysis

The program is designed to be used by a user supplying the location as input for the files to be compared and the location where all the output should be written, which will be in Microsoft’s Excel 2003 format. Once the desired files have been highlighted and the output file set, running the application reads each file into memory. This takes O(m*n) time where m is the number of files to be read and n is the number spots present on the given arrays.

The program calculates the means, standard deviation, and coefficient of variation by going through each key within the GenePix file. The minima, maxima, and median values are found by using the native sort in Perl, a quicksort of complexity O(n log n) that offsets the possibility of a worst case scenario by shuffling the input. The algorithm runs on all controls, then all peptide data.

Conclusion

There are diverse possibilities in the field of microarray production, experimental design, and analysis. The methods have grown and have been refined in a relatively short amount of time, and microarrays have introduced one of the major problems the field of bioinformatics arose to solve: how to efficiently handle an abundance of data.

Microarrays can be designed to accommodate various biological matrices, but are well known for their use in nucleotide and amino acid research. Nucleotide arrays can use cDNA or short oligonucleotides and be measured as comparisons of different tissue states or singly, on a single-channel array.

There are many tools used to analyze microarrays, and more are being developed and distributed, each with their own strengths and weaknesses. The steps involved in analysis begin with the acquisition and normalization of data, then to the preprocessing of data by the removal of outliers and poor signals as well as the imputation of missing data. After processing, the data can be further analyzed to make a variety of inferences from differential gene expression to gene-relatedness through clustering.

This program was developed to be used after the normalization of data, during the pre-processing step to quickly and easily compute the summary statistics necessary to make decisions about what data may be of interest and what data to discard.

References

Draghici, Sorin. 2012. Statistics and Data Analysis for Microarrays Using R and Bioconductor. Second Edition. Florida, Boca Raton: CRC Press

Watson, J. D., Baker, T. A., Bell, S. P., Gann, A., Levine,M., Losick, R., Harrison, S. C. 2013. Molecular Biology of the Gene. Seventh Edition. New York, Cold Spring Harbor: Cold Spring Harbor Press

Li, S., Wang, W., Fan, K., Yang, K., 2015. Genome-Wide Identification and Characterization of Reference Genes with Different Transcript Abundances for *Streptomyces coelicolor*. Scientific Report 5:15840 DOI: 10.1038/srep15840

Hruz, T., Wyss, M., Melene, D., Pfaffl, M. W., Masanetz, S., Borghi, L., Verbrugghe, P., … Zimmermann, P. 2011. RefGenes: Identification of Reliable and Condition Specific Reference Genes for RT-qPCR Data Normalization. BMC Genomics 12:156

Dufva, M., 2005. Fabrication of High Quality Microarrays. Biomolecular Engineering. doi:10.1016/j.bioeng.2005.09.003

Barbulovic-Nad, I., Lucenta, M., Sun, Y., Zhang, M., Wheeler, A. R., Bussmann, M. 2006. Bio-Microarray Fabrication Techniques-A Review. Critical Reviews in Biotechnology 26:4, 237-259. doi:10.1080/07388550600978358

Figueroa, A., Tsai, P., Bent, E., Guo, R. 2008. Robust Spot Finding in Microarray Images with Distortions. 30th Annual International IEEE EMBS Conference. Vancouver, British Columbia, Canada. August 20-24, 2008. 978-1-4244-1815-2/08/

Agilent Technologies. 2015. Agilent Microarray Technology. 2015. Retrieved from: http://www.genomics.agilent.com/article.jsp?pageId=2011

Affymetrix. 2015. GeneChip Microarray Curriculum: GeneChip Microarrays. Retrieved from: http://www.affymetrix.com/estore/about_affymetrix/outreach/educator/curricula.affx

Tomita, J., Ueno, T., Mitsuyoshi, M., Kuma, S., Kuma, K. 2015. The NMDA Receptor Promotes Sleep in the Fruit Fly, *Drosophila melanogaster*. PLoS ONE 10(5):e0128101. DOI:10.1371/journal.pone.0128101

Amersham Biosciences. 2003. Stability Studies of Dyes in Microarray Applications. Retrieved from: https://www.gelifesciences.com/gehcls_images/GELS/Related%20Content/Files/1314735988470/litdoc63005239_20110830234930.pdf

Yang, Y. H., Duduit, S. 2015. Normalization: Bioconductor's Marray Package. R package version 1.48.0. http://www.maths.usyd.edu.au/u/jeany/

Park, T., Yi, S., Kang, S., Lee, S., Lee, Y., Simon, R. 2003. Evaluation of Normalization Methods for Microarray Data. BMC Bioinformatics 4:33. http://www.biomedcentral.com/1471-2105/4/33

Irizarray, R. A., Bolstad, B. M., Collin, F., Cope, L. M., Hobbs, B., Speed, T. P. 2003. Summaries of Affymetrix GeneChip Probe Level Data. Nucleic Acids Research, Vol 31, No 4, e15. DOI: 10.1093/nar/gng015

Pepper, S. D., Saunders, E. K., Edwards, L. E., Wilson, C. L., Miller, C. J. 2007. The Utility of MAS5 Expression Summary and Detection Call Algorithms. BMC Bioinformatics, 8:273. doi:10.1186/1471-2105-8-273

Appendix: Program Code

```
#!/bin/env perl
use strict;
use Wx;

#use PAR::Kludge;

my $wx = QCPC::MainWindow->new();
$wx->MainLoop;

package QCPC::MainWindow;
use strict;
use base qw(Wx::App);

sub OnInit {
        my $self = shift;
        my $frame =
          QCPC::Frame->new( undef, -1, "QCPC Application", [ 1, 1 ], [ 400, 400 ] );
        $self->SetTopWindow($frame);
        $frame->Show(1);
}
1;

package QCPC::Frame;
use strict;
use File::Find;
use Math::Complex;
use File::Basename;
use Excel::Writer::XLSX;
use Excel::Writer::XLSX::Utility;
use List::Util qw(max min);

use base qw(Wx::Frame);

use Wx::Event qw( EVT_BUTTON );

my @control_list;
my @desired_list;


my @gprs;
my @graphs;    # holds all of the various graphs created for the excel file for
               # output at the end of the run
my %gprs_stats_cache
  ;            # cache the calculated means and stddev for each of the gpr files
my %peptide_row;    # holds the rows for peptides that are output
my ( $row, $col ) = ( 0, 0 );

my @mean;
my ( $datasheet, $graph );
my @peptides;

sub new {
        my $class = shift;
        my $self  = $class->SUPER::new(@_);    # call the superclass' constructor
            # Then define a Panel to put the button on
        my $panel = Wx::Panel->new(
                $self,    # parent
                -1        # id
        );
        my ( $gpr_dir_button, $xls_button, $run_button ) = ( 1 .. 10 );
        $self->{default_columns} =
          [ "name", "id", "f532 median", "f532 mean", "b532 median", "b532 mean" ];

        $self->{list} = Wx::ListBox->new(
                $panel, -1,
                [ 0,   180 ],
                [ 240, 100 ],
                [], &Wx::wxLB_EXTENDED
        );
        @{ $self->{filter} } = (
                Wx::RadioButton->new(
                        $panel, -1,
                        "Mean", [ 250, 60 ],
                        &Wx::wxDefaultSize, &Wx::wxRB_GROUP
                ),

                Wx::RadioButton->new(
                        $panel, -1, "Median", [ 250, 80 ],
                        &Wx::wxDefaultSize
                )
        );

        $self->{gpr} = Wx::Button->new(
                $panel,             # parent
                $gpr_dir_button,    # ButtonID
                "GPR Folder",       # label
                [ 0, 300 ]          # position
        );
        EVT_BUTTON(
                $self,              # Object to bind to
                $gpr_dir_button,    # ButtonID
                \&GPRClicked        # Subroutine to execute
        );
        $self->{xls} = Wx::Button->new(
                $panel,             # parent
                $xls_button,        # ButtonID
                "XLS File",         # label
                [ 80, 300 ]         # position
        );
        EVT_BUTTON(
                $self,              # Object to bind to
                $xls_button,        # ButtonID
                \&XLSClicked        # Subroutine to execute
        );
        $self->{run} = Wx::Button->new(
                $panel,             # parent
                $run_button,        # ButtonID
                "Run Analysis",     # label
                [ 160, 300 ]        # position
```

```
);
EVT_BUTTON(
        $self,          # Object to bind to
        $run_button,    # ButtonID
        \&RunClicked    # Subroutine to execute
);

Wx::StaticText->new(
        $panel,         # parent
        1,              # id
        "Controls",     # label
        [ 0, 3 ]        # position
);

$self->{controls} = Wx::TextCtrl->new(
        $panel, -1,
        "fiducial, empty, blank, negative, positive, landing_lights",
        [ 80,  0 ],
        [ 160, 20 ],
);

Wx::StaticText->new(
        $panel,             # parent
        1,                  # id
        "Columns to Use",   # label
        [ 0, 60 ]           # position
);

$self->{columns} = Wx::ListBox->new(
        $panel, -1,
        [ 80,  60 ],
        [ 160, 100 ],
        [], &Wx::wxLB_EXTENDED
);
$self->{columns}->Set( $self->{default_columns} );
foreach ( 0 .. scalar @{ $self->{default_columns} } ) {
```

```
                $self->{columns}->SetSelection($_);
        }
        return $self;
}

=head1 C<file_peptide_stats>

file_peptide_stats uses the cached gpr data held in @gprs to calculate the
mean, standard deviation and cv of each array for the non-controls.

        file_peptide_stats()

=cut

sub file_peptide_stats {
        my ($self) = @_;
        foreach my $gpr (@gprs) {
                my @data = ( $gpr->{"file"}, "Signal" );
                foreach my $signal (@mean) {
                        my %data;
                        my @signals;
                        my $mean;
                        my $stddev;
                        foreach my $key ( keys %{ $gpr->{"peptides"} } ) {
                                $mean += $gpr->{"peptides"}->{$key}->{$signal};
                                push @signals, $gpr->{"peptides"}->{$key}->{$signal};
                        }
                        $mean /= scalar( keys %{ $gpr->{"peptides"} } );
                        push @data, "$mean";

                        foreach my $key ( keys %{ $gpr->{"peptides"} } ) {
                                $stddev += ( $gpr->{"peptides"}->{$key}->{$signal} - $mean )**2;
                        }
                        $stddev = sqrt( $stddev / scalar( keys %{ $gpr->{"peptides"} } ) );
                        push @data, "$stddev";
                        my $cv = $stddev / $mean;
                        push @data, "$cv";

                        # min and max of the peptides
                        my ($min) = sort { $a <=> $b } @signals;
                        my ($max) = sort { $b <=> $a } @signals;
                        push @data, ( "$min", "$max" );

                        $data{"mean"}                = $mean;
                        $data{"stddev"}              = $stddev;
                        $gprs_stats_cache{ $gpr->{"file"} . "-" . $signal } = \%data;
                }
                $self->{worksheet}->write( $row, $col, \@data );
                $row++;
        }
}

=head1 C<file_control_stats>
file_control_stats prints out the mean, standard deviation and cv for the
different controls found in the gpr files

        file_control_stats()

=cut

sub file_control_stats {
        my ($self) = @_;
        foreach my $gpr (@gprs) {
                foreach my $control ( keys %{ $gpr->{"controls"} } ) {
                        my @data = ( $gpr->{"file"}, "$control" );
                        foreach my $signal (@mean) {
                                my $mean;
                                my $stddev;
                                foreach my $key ( @{ $gpr->{"controls"}->{$control} } ) {
                                        $mean += $key->{$signal};
                                }
                                $mean /= scalar( @{ $gpr->{"controls"}->{$control} } );
                                push @data, "$mean";

                                foreach my $key ( @{ $gpr->{"controls"}->{$control} } ) {
                                        $stddev += ( $key->{$signal} - $mean )**2;
                                }
                                $stddev = sqrt(
                                        $stddev / scalar( @{ $gpr->{"controls"}->{$control} } ) );
                                push @data, "$stddev";
                                my $cv = $stddev / $mean;
                                push @data, "$cv";

                                # min and max of the controls
                                my ($min) =
                                  sort { $a->{$signal} <=> $b->{$signal} }
                                  @{ $gpr->{"controls"}->{$control} };
                                my ($max) =
                                  sort { $b->{$signal} <=> $a->{$signal} }
                                  @{ $gpr->{"controls"}->{$control} };
                                push @data, ( $min->{$signal}, $max->{$signal} );
                        }
                        $self->{worksheet}->write( $row, $col, \@data );
                        $row++;
                }
        }
}

=head1 C<file_total_stats>

file_total_stats uses the cached gpr data held in @gprs to calculate the total
inter-array mean, standard deviation and cv for all the peptides and all the
controls separately.

        file_total_stats()

=cut
```

```
sub file_total_stats {
        my ($self) = @_;
        my @controls;    # the controls that were actually found for the first file
                  #(and so for each file after I hope)
        $col = 1;
        foreach my $signal (@mean) {
                $row = 1;
                foreach my $control (@control_list) {
                        my $mean;
                        my $stddev;
                        my $count;
                        next
                          if ( !defined( $gprs[0]->{"controls"}->{$control} ) );
                        push @controls, $control;

                        foreach my $gpr (@gprs) {
                                foreach my $key ( @{ $gpr->{"controls"}->{$control} } ) {
                                        $mean += $key->{$signal};
                                        $count++;
                                }
                        }
                        next if ( !$count );
                        $mean /= $count;

                        foreach my $gpr (@gprs) {
                                foreach my $key ( @{ $gpr->{"controls"}->{$control} } ) {
                                        $stddev += ( $key->{$signal} - $mean )**2;
                                }
                                $stddev = sqrt( $stddev / $count );
                        }
                        my $cv = $stddev / $mean;
                        $self->{worksheet}
                          ->write( $row++, $col, [ "$mean", "$stddev", "$cv" ] );
                }

                # found the mean for the various control types. Now do the same for all
                # the peptides
                my ( $mean, $stddev, $cv, $count );

                foreach my $gpr (@gprs) {
                        foreach my $key ( keys %{ $gpr->{"peptides"} } ) {
                                $mean += $gpr->{"peptides"}->{$key}->{$signal};
                                $count++;
                        }
                }
                next if ( !$count );
                $mean /= $count;

                foreach my $gpr (@gprs) {
                        foreach my $key ( keys %{ $gpr->{"peptides"} } ) {
                                $stddev += ( $gpr->{"peptides"}->{$key}->{$signal} - $mean )**2;
                        }
                        $stddev = sqrt( $stddev / $count );
                }
                $cv = $stddev / $mean;
                $self->{worksheet}
                  ->write( $row++, $col, [ "$mean", "$stddev", "$cv" ] );
                $col += 3;
        }
        $col = 0;
        $row = 1;
        foreach ( unique(@controls) ) {
                $self->{worksheet}->write( $row++, $col, $_ );
        }
        $self->{worksheet}->write( $row, $col, "Signal" );
}

=head1 C<col_to_ref>

Converts A1 excel notation to $A$1

        col_to_ref($col)
```

```
=cut

sub col_to_ref {
        my ($col) = @_;
        if ( $col =~ /([a-z]+)([\d]+)/i ) { $col = '$' . $1 . '$' . $2; }
        return $col;
}

sub log_base {
        my ( $base, $value ) = @_;
        if ( $value > 0 && $base > 0 ) {
                return log($value) / log($base);
        }
        else { return 0; }
}

sub GPRClicked {
        my ( $self, $event ) = @_;
        $self->{dir} = Wx::DirSelector("Select GPR Directory");
        if ( -e $self->{dir} && -d $self->{dir} ) {
                opendir my $dh, $self->{dir}
                  || die "can't opendir " . $self->{dir} . " $!";
                my @files = grep { /\.gpr$/ && -f $self->{dir} . "/$_" } readdir($dh);
                $self->{list}->Set( \@files );
                open my $fh, $self->{dir} . "/" . $files[0];
                while ( my $line = <$fh> ) {
                        if ( $line =~ /^"?block"?/i ) {
                                chomp($line);
                                $line =~ s/["']//g;

                                my @current_selections = $self->{columns}->GetSelections();
                                foreach ( 0 .. $#current_selections ) {
                                        $current_selections[$_] =
                                          $self->{columns}->GetString( $current_selections[$_] );
                                }
                        #split the line on tabs for mapping
                        my @temp = split( /\t/, lc($line) );
                        $self->{columns}->Set( \@temp );
                        foreach ( 0 .. $#temp ) {
                            if ( inlist( $temp[$_], @current_selections ) ) {
                                $self->{columns}->SetSelection($_);
                            }
                        }
                        last;
                    }
                }

        }
}

sub XLSClicked {
        my ( $self, $event ) = @_;
        ( $self->{xls} ) = Wx::FileSelector("Select XLSX File for writing");
        if ( $self->{xls} !~ /\.xlsx$/i ) {
                $self->{xls} =~ s/\.[a-z]{2,4}$//i;
                $self->{xls} .= '.xlsx';
        }
}

sub RunClicked {
        my ( $self, $event ) = @_;
        @control_list = @desired_list = @gprs = @graphs = @peptides = ();

        %gprs_stats_cache = %peptide_row = {};
        $row            = 0;
        $col            = 0;

        if ( $self->{dir} eq "" ) {
                Wx::MessageBox( "No Directory was Selected", "ERROR!!!", "", $self );
                return;
```

```
}
if ( ref( $self->{xls} ) ne "" || $self->{xls} eq "" ) {
        Wx::MessageBox( "No Excel File was Selected", "ERROR!!!", "", $self );
        return;
}
@control_list = split( /\s?+,\s?+/, $self->{controls}->GetValue );
@desired_list = $self->{columns}->GetSelections();

foreach ( 0 .. $#desired_list ) {
        $desired_list[$_] = $self->{columns}->GetString( $desired_list[$_] );
}

my $filter = "median";
foreach my $rb ( @{ $self->{filter} } ) {
        if ( $rb->GetValue() ) {
                $filter = $rb->GetLabel();
        }
}

my $search = '^[fb]\d{1,3}.*' . lc($filter) . '$';

@mean = grep( /$search/, @desired_list );

my $EXCEL = Excel::Writer::XLSX->new( $self->{xls} );

my @files = $self->{list}->GetSelections();
foreach ( 0 .. $#files ) {
        files( $self->{dir} . "/" . $self->{list}->GetString( $files[$_] ) );
}

$self->{worksheet} = $EXCEL->add_worksheet('Files');
my $graphsheet = $EXCEL->add_worksheet('Graphs');

foreach my $gpr (@gprs) {
        $self->{worksheet}->write( $row, $col++, $gpr->{"file"} );
        foreach ( keys %{ $gpr->{controls} } ) {
```

```
                    $self->{worksheet}
                      ->write( $row, $col, [ $_, scalar @{ $gpr->{controls}->{$_} } ] );
                    $col += 2;
          }
          $self->{worksheet}->write( $row, $col,
                    [ "peptides", scalar keys( %{ $gpr->{peptides} } ) ] );
          $row++;
          $col = 0;
}

$self->{worksheet} = $EXCEL->add_worksheet('File Statistics');

my @headers = ( "File", "Control" );
foreach (@mean) {
          push @headers, ( "$_", "$_ stddev", "$_ cv", "Min", "Max" );
}
$self->{worksheet}->write( 0, 0, \@headers );
$row = 1;
$col = 0;

$self->file_peptide_stats();
$self->file_control_stats();

$self->{worksheet} = $EXCEL->add_worksheet('Array Data');
$self->{worksheet}->hide();
$row = 1;
$col = 0;

my $count = 1;
my @peptides;
foreach my $gpr (@gprs) {
          push @peptides, keys %{ $gpr->{"peptides"} };
}
@peptides = unique(@peptides);
for ( my $i = 0 ;
          $i < @peptides ; $i += max( 1, int( @peptides / 20000 ) ) )
```

```
{
        $self->{worksheet}->write( $count, $col, $peptides[$i] );
        $peptide_row{ $peptides[$i] } = $count++;
}
$col = 1;

# Write out the array data to an excel sheet then create the relation graphs
# for each pairs of data written out. Think I'll merge these into each other
# for simplicity of the creation of the graphs.
my ($mean) = @mean;
my @keys = keys %peptide_row;
foreach my $gpr (@gprs) {
        $self->{worksheet}->write( 0, $col, $gpr->{'file'} );
        foreach my $key (@keys) {
                if (   defined( $gpr->{"peptides"}->{$key} )
                        && defined( $peptide_row{$key} ) )
                {
                        $self->{worksheet}->write( $peptide_row{$key}, $col,
                                log_base( 10, $gpr->{"peptides"}->{$key}->{$mean} ) );
                }
        }
        $col++;
}
foreach my $i ( 1 .. $col - 2 ) {
        foreach my $j ( $i + 1 .. $col - 1 ) {
                my $graph = $EXCEL->add_chart( type => 'scatter', embedded => 1 );
                $graph->set_title( name => $gprs[ $i - 1 ]->{'file'} . ' vs '
                          . $gprs[ $j - 1 ]->{"file"} );
                $graph->add_series(
                        categories => '='
                          . $self->{worksheet}->get_name() . '!'
                          . col_to_ref( xl_rowcol_to_cell( 1,           $i ) ) . ':'
                          . col_to_ref( xl_rowcol_to_cell( scalar @keys, $i ) ),
                        values => '='
                          . $self->{worksheet}->get_name() . '!'
                          . col_to_ref( xl_rowcol_to_cell( 1,           $j ) ) . ':'
```

```
					. col_to_ref( xl_rowcol_to_cell( scalar @keys, $j ) ),
				);
				$graph->set_x_axis( name => "log10 " . $gprs[ $i - 1 ]->{"file"} );
				$graph->set_y_axis( name => "log10 " . $gprs[ $j - 1 ]->{"file"} );
				$graph->set_legend( position => 'none' );
				push @graphs, $graph;
		}
}

$datasheet = $EXCEL->add_worksheet('Log Ratio Data');
$self->{worksheet} = $EXCEL->add_worksheet('File Log Ratios');
$datasheet->hide();
$row = $col = 0;

my $graph = $EXCEL->add_chart(
		type     => 'column',
		embedded => 1
);

my $data_col = 0;
my %ratio_buckets;	# gather up all the log2 ratio data into buckets
foreach ( keys %peptide_row ) {
		$datasheet->write( $peptide_row{$_}, $data_col, $_ );
}
$data_col++;
$row = $col = 0;
($mean) = @mean;
$self->{worksheet}->write( $row++, $col,
		[ "File 1", "File 2", "$_ Ratio", "$mean 95% CI" ] );
foreach my $i ( 0 .. $#gprs - 1 ) {
		my $gpr1 = $gprs[$i];
		my @keys = keys %{ $gpr1->{"peptides"} };
		foreach my $j ( $i + 1 .. $#gprs ) {
				my @CI;	# confidence interval
				my $count = 0;
				my $array_mean;
my $gpr2 = $gprs[$j];
$datasheet->write( 0, $data_col,
        $gpr1->{"file"} . "/" . $gpr2->{"file"} );
foreach my $key (@keys) {
        if (   defined( $gpr1->{"peptides"}->{$key} )
                && defined( $gpr2->{"peptides"}->{$key} ) )
        {
                if (   $gpr1->{"peptides"}->{$key}->{$mean} > 0
                        && $gpr2->{"peptides"}->{$key}->{$mean} > 0 )
                {
                        $count++;
                        push @CI,
                          log_base( 2,
                                $gpr1->{"peptides"}->{$key}->{$mean} /
                                  $gpr2->{"peptides"}->{$key}->{$mean} );
                        $ratio_buckets{ int( $CI[$#CI] * 10 ) }++;
                        $CI[$#CI] = abs( $CI[$#CI] );
                        $array_mean += $CI[$#CI];
                        if ( defined( $peptide_row{$key} ) ) {
                                $datasheet->write( $peptide_row{$key}, $data_col,
                                        $CI[$#CI] );
                        }

                }
        }
}
if ( $count > 0 ) {
        @CI = sort @CI;
        $self->{worksheet}->write(
                $row++,
                $col,
                [
                        $gpr1->{"file"},      $gpr2->{"file"},
                        $array_mean / $count, 2**$CI[ int( .95 * @CI ) ]
                ]
        );
```

```
                }
                $data_col++;
        }
}
my ( $x_name, $y_name );
$x_name = $y_name = '=' . $datasheet->get_name() . '!';
$data_col++
  ; # create an empty column between the file ratios and the overall bucket data
$row = 0;

$datasheet->write( $row++, $data_col, [ "Power", "Count" ] );
$x_name .= col_to_ref( xl_rowcol_to_cell( $row, $data_col ) ) . ':';
$y_name .= col_to_ref( xl_rowcol_to_cell( $row, $data_col + 1 ) ) . ':';
foreach my $key ( sort { $a <=> $b } keys %ratio_buckets ) {
        $datasheet->write( $row++, $data_col,
                [ $key / 10, $ratio_buckets{$key} ] );
}
$x_name .= col_to_ref( xl_rowcol_to_cell( $row - 1, $data_col ) );
$y_name .= col_to_ref( xl_rowcol_to_cell( $row - 1, $data_col + 1 ) );
$graph->set_title( name => 'Log Ratio Buckets' );
$graph->set_x_axis( name => 'Log2 Power' );
$graph->set_y_axis( name => 'Count' );
$graph->add_series(
        name       => 'Ratios',
        categories => $x_name,
        values     => $y_name,
);
push @graphs, $graph;

$self->{worksheet} = $EXCEL->add_worksheet('Across Array Stats');
@headers = ("Control");
foreach (@mean) {
        push @headers, ( "$_", "$_ stddev", "$_ cv" );
}
$self->{worksheet}->write( 0, 0, \@headers );
$row = 1;
```

```
$col = 0;
$self->file_total_stats();

$self->{worksheet} = $EXCEL->add_worksheet('Correlation');
$col = $row = 0;

# Correlation calculations

$self->{worksheet}->write( $row++, $col,
        [ "File 1", "File 2", "Correlation", "R-squared" ] );
($mean) = @mean;

#print $mean. "\n";
foreach my $i ( 0 .. $#gprs - 1 ) {
        my $gpr1 = $gprs[$i];
        my @keys = keys %peptide_row;
        foreach my $j ( $i + 1 .. $#gprs ) {
                my $count = 0;
                my $r;
                my $gpr2 = $gprs[$j];
                foreach my $key (@peptides) {
                        if (   defined( $gpr1->{"peptides"}->{$key} )
                                && defined( $gpr2->{"peptides"}->{$key} ) )
                        {
                                if (   $gpr1->{"peptides"}->{$key}->{$mean} >= 0
                                        && $gpr2->{"peptides"}->{$key}->{$mean} >= 0 )
                                {
                                        $count++;
                                        $r += (
                                                (
                                                        $gpr1->{"peptides"}->{$key}->{$mean} -
                                                          $gprs_stats_cache{ $gpr1->{"file"} . "-"
                                                                  . $mean }->{"mean"}
                                                ) /
                                                  $gprs_stats_cache{ $gpr1->{"file"} . "-" . $mean }
                                                  ->{"stddev"}
                                        ) * (
                                              (
                                                    $gpr2->{"peptides"}->{$key}->{$mean} -
                                                     $gprs_stats_cache{ $gpr2->{"file"} . "-"
                                                           . $mean }->{"mean"}
                                              ) /
                                               $gprs_stats_cache{ $gpr2->{"file"} . "-" . $mean }
                                               ->{"stddev"}
                                        );
                                  }
                            }
                      }

                      $r /= $count if ( $count > 0 );

                      $self->{worksheet}->write(
                            $row++, $col,
                            [ $gpr1->{"file"}, $gpr2->{"file"}, $r, $r**2, $mean ]
                      );
                }
          }

          $self->{worksheet} = $EXCEL->add_worksheet('Slide Density Data');
          $self->{worksheet}->hide();
          my $max = 0;   # what is the max array size so that the index can be created
          my $col = 1;
          foreach my $i ( 0 .. $#gprs ) {
                my $gpr = $gprs[$i];
                foreach my $j ( 0 .. $#mean ) {
                      my $mean = $mean[$j];
                      $self->{worksheet}->write( 0, $col, $gpr->{"file"} . " " . $mean );
                      my @density;
                      foreach my $key ( keys %{ $gpr->{"peptides"} } ) {
                            $density[
                             int(
                                   10 * log_base( 10, $gpr->{"peptides"}->{$key}->{$mean} ) )
                            ]++;
                      }
                      $self->{worksheet}->write( 1, $col, [ \@density ] );
                      $max = max( $max, scalar $#density );
                      my $graph = $EXCEL->add_chart( type => 'column', embedded => 1 );
                      my ( $x_name, $y_name );
                      $x_name = $y_name = '=' . $self->{worksheet}->get_name() . '!';
                      $x_name .=
                        col_to_ref( xl_rowcol_to_cell( 1,    0 ) ) . ':'
                       . col_to_ref( xl_rowcol_to_cell( $max, 0 ) );
                      $y_name .=
                        col_to_ref( xl_rowcol_to_cell( 1,    $col ) ) . ':'
                       . col_to_ref( xl_rowcol_to_cell( $max, $col ) );

                      $graph->set_title( name => $gpr->{"file"} . " " . $mean );
                      $graph->set_x_axis( name => 'Log10 Signal' );
                      $graph->set_y_axis( name => 'Count' );
                      $graph->add_series(
                            name       => $gpr->{"file"} . " " . $mean,
                            categories => $x_name,
                            values     => $y_name,
                      );
                      push @graphs, $graph;
                      $col++;
                }
          }
          my $i = 2;
          ($mean) = @mean;
          foreach my $gpr (@gprs) {
                foreach my $control ( keys %{ $gpr->{"controls"} } ) {
                      my @density;
                      $self->{worksheet}
                       ->write( 0, $col, $gpr->{"file"} . " " . $control );
                      foreach ( @{ $gpr->{"controls"}->{$control} } ) {
                            $density[ int( 10 * log_base( 10, $_->{$mean} ) ) ]++;
                      }
                $max = max( $max, scalar $#density );
                $self->{worksheet}->write( 1, $col, [ \@density ] );
                my $graph = $EXCEL->add_chart(
                            name     => "A$i",
                            type     => 'column',
                            embedded => 1
                );
                my ( $x_name, $y_name );
                $x_name = $y_name = '=' . $self->{worksheet}->get_name() . '!';
                $x_name .=
                      col_to_ref( xl_rowcol_to_cell( 1,    0 ) ) . ':'
                    . col_to_ref( xl_rowcol_to_cell( $max, 0 ) );
                $y_name .=
                      col_to_ref( xl_rowcol_to_cell( 1,    $col ) ) . ':'
                    . col_to_ref( xl_rowcol_to_cell( $max, $col ) );

                $graph->set_title( name => $gpr->{"file"} . " " . $control );
                $graph->set_x_axis( name => 'Log10 Signal' );
                $graph->set_y_axis( name => 'Count' );
                $graph->add_series(
                            name       => $gpr->{"file"} . " " . $control,
                            categories => $x_name,
                            values     => $y_name,
                );
                push @graphs, $graph;
                $col++;
            }
          }
          foreach ( 0 .. $max ) {
            $self->{worksheet}->write( $_ + 1, 0, $_ / 10 );
          }
          foreach ( 0 .. $#graphs ) {
            $graphsheet->insert_chart(
                xl_rowcol_to_cell( 20 * int( $_ / 5 ), 9 * ( $_ % 5 ) ),
                $graphs[$_] );
          }
```

```
}

=head1 C<inlist>

inlist searches through a list of items to see if there is a match with the
desired item.  Handles dealing with an array of strings or an array of sequence
objects from BioPerl.  This function could probably be made more robust, but
nothing has forced it to be changed any further.

        inlist($item, @find_in_array)

=cut

sub inlist {
        my $value = shift;
        $value =~ s/(["'\\\[\]])/\\$1/g;
        local ($_);
        if ( ref( $_[0] ) eq 'HASH' ) {
                $_ = '~';
                foreach my $seq (@_) {
                        $_ .= $seq->seq . '~';
                }
        }
        elsif ( !( ref( $_[0] ) ) ) {
                $_ = '~' . join( '~', @_ ) . '~';
        }
        return (0) unless /^(.*?~$value)~/i;
        my $chunk = $1;
        $chunk =~ tr/~//;
}

=head1 C<average>
average returns the mean value of an array of values. They do not need to be
presorted before calling.

        average(\@values)
```

```
=cut

sub average {
	my ($array_ref) = @_;
	my $sum;
	my $count = scalar @$array_ref;
	foreach (@$array_ref) { $sum += $_; }
	return $sum / $count;
}

=head1 C<median>
median returns the median value of an array of values. They do not need to be
presorted before calling.

	median(\@values)

=cut

sub median {
	@_ == 1 or die('Sub usage: $median = median(\@array);');
	my ($array_ref) = @_;
	my $count = scalar @$array_ref;

	# Sort a COPY of the array, leaving the original untouched
	my @array = sort { $a <=> $b } @$array_ref;
	if ( $count % 2 ) {
		return $array[ int( $count / 2 ) ];
	}
	else {
		return ( $array[ $count / 2 ] + $array[ $count / 2 - 1 ] ) / 2;
	}
}

=head1 C<unique>
```

```
Takes in a list of items and returns an array of just the unique items contained in it.
        unique(@items)

=cut

sub unique {
        my @list  = @_;
        my %seen  = ();
        my @uniqu = grep { !$seen{$_}++ } @list;
        return @uniqu;
}

=head1 C<files>
files is called by the File::Find module to compile together an array of the
gpr files for future processing

        files()

=cut

sub files {
        $_ = shift @_;

        #print "$_\n";
        return if ( $_ !~ /\.gpr$/ );

        my %gpr;

        my %headers;

        my %peptides
          ;    # this should hold the data for the various peptides on an array
        my %controls
          ;    # this should hold the data for the various controls on an array

        open my $fh, $_ || return;
```

```
	$gpr{'file'} = fileparse( $_, '.gpr' );

	# need to strip the header rows off the GPR file
	# probably not important to keep them around

	while ( my $line = <$fh> ) {
		if ( $line =~ /^"?block"?/i ) {

			# newline character was causing name issues, but wasn't noticed
			# with genepix gpr files as the name column wasn't the last one in
			# the list for the header row
			chomp($line);
			$line =~ s/["']//g;

			#split the line on tabs for mapping
			my @temp = split( /\t/, $line );
			foreach ( 0 .. $#temp ) {
				$headers{ lc( $temp[$_] ) } = $_;

				#print lc( $temp[$_] ) ."\t".$_."\n";
			}
			last;
		}
	}

# at this point we've read in all the various header rows and setup the column
# header data now we need to parse through the actual data columns and pull out
# the data that we need

	while ( my $line = <$fh> ) {
		my %datapoint;
		chomp $line;
		$line =~ s/["']//g;
		my @line = split( /\t/, $line );

		foreach (@desired_list) {
```

```
                $datapoint{$_} = $line[ $headers{$_} ];
        }

        # is this a control element or a data point. each is handled slightly
        # differently.
        if (   inlist( $line[ $headers{'name'} ], @control_list )
                || inlist( $line[ $headers{'id'} ], @control_list ) )
        {
                if ( $line[ $headers{'name'} ] =~ /[\w\d]+/i ) {
                        push @{ $controls{ lc( $line[ $headers{'name'} ] ) } },
                          \%datapoint;
                }
                else {
                        push @{ $controls{ lc( $line[ $headers{'id'} ] ) } },
                          \%datapoint;
                }

                #print "Control Name: " . $line[ $headers{'name'} ] . "\n";
        }

  # else this is a piece of data. Need to temporarily store these to do
  # data manipulation on a single array of data before comparing data across
  # the various gpr files.
        else {
                push @{ $peptides{ $line[ $headers{'name'} ] } }, \%datapoint;

                #print "Peptide Name: ".$line[ $headers{'name'} ]."\n";
        }
}

my %temp;

foreach ( keys %peptides ) {
        if ( @{ $peptides{$_} } <= 1 ) {
                $temp{$_} = ${ $peptides{$_} }[0];
        }
```

```
            else {
                my %datapoint;
                foreach my $element (@desired_list) {
                    if ( $element eq "name" ) {
                        $datapoint{$element} = $_;
                        next;
                    }
                    if ( $element eq "id" ) {
                        $datapoint{$element} = ${ $peptides{$_} }[0]->{$element};
                        next;
                    }
                    my @data
                      ; # holds the values that are going to be either averaged or median
                    foreach my $data ( @{ $peptides{$_} } ) {
                        push @data, $data->{$element};
                    }
                    if ( $element =~ /median/i ) {
                        $datapoint{$element} = median( \@data );
                    }
                    elsif ( $element =~ /mean/i ) {
                        $datapoint{$element} = average( \@data );
                    }
                }
                $temp{$_} = \%datapoint;
            }

        }
        %peptides      = %temp;
        $gpr{"controls"} = \%controls;
        $gpr{"peptides"} = \%peptides;
        push @gprs, \%gpr;
}
1;
```