Model Class

The model class stores information about the models (lines) we are searching, such as its slope, intercept, attached points, etc. Since almost all of its information needs to be carefully managed, all its member are held private, but with a lot of methods to interact indirectly with them. You will notice two strange members in the class, being “energy” and “fitness”. Energy is the sum of every point attached to the model and the line that it stores, this is made because PEARL and RUBY use this information to see how good a set of models is. Fitness is a measure to quantify how good or bad a single model is, using $fitness = \frac{b^2 + 10 * Energy}{n^2 * p}$ where ‘b’ is the line intercept, ‘n’ is the number of point in the model and ‘p’ the number of parallel models. This function was defined experimentally and seems to work quite well, the idea behind it being to search the model that has the smallest fitness. Since we are searching for the smallest ‘b’ and energy, and the model that has the most number of points and parallel models, this fitness function generates a value that quantify all these parameters, and put a “weight” into the intercept and number of points, the values that are most important. The energy is multiplied by 10 to make it somewhat competitive, as intercept values are often way bigger.

Public Static Funcitons

- ## linearFit( )

  Model Model::linearFit (const std::vector<Point> & vec)

  Calculates the linear regression for the points inside the vector ‘vec’.

  Parameters

  vec is the vector of points

Public Methods

- ## Model( )

  Model::Model (const double aa = 10^{20},
  const double bb = 10^{20}
)

  Contructor that assign slope and intercept for a model, or the default values

  Parameters

  aa is the line’s slope
  bb is the line’s intercept

◆ findBestModel( ) [1/2]

void Model::findBestModel ( )

Using the points in the model and linearFit to find the best model to this set of points

◆ findBestModel( ) [2/2]

void Model::findBestModel (const std::vector<Point> & vec)

Assign the points in “vec” to the model, then calculates the best model using linearFit

Parameters

__vec__ is the set of points to be assigned

◆ getSlope( )

double Model::getSlope ( ) const

Gets the slope of the model

◆ getIntercept( )

double Model::getIntercept ( ) const

Gets the intercept of the model

◆ getEnergy( )

double Model::getEnergy ( ) const

Gets the energy of the model

◆ getParallelCount( )

int Model::getParallelCount ( ) const

Gets the parallel count of the model

◆ getFitness( )

double Model::getFitness ( ) const

Gets model’s fitness

◆ getPositivePointsNum( )

int Model::getPositivePointsNum ( ) const

Gets the number of positive points in model

- ### getPointsSize( )

  int Model::getPointsSize ( ) const

  Gets the number of points in model

- ### getFirstAndLastPoint( )

  std::pair<Point, Point> Model::getFirstAndLastPoint ( ) const

  Gets first the closest point to the origin and second the farthest point, using the x-coordinate

- ### getPointsInModel( )

  std::vector<Point> Model::getPointsInModel ( ) const

  Gets the set of points attached to this model

- ### getPointsVecBegin( )

  std::vector<Point>::const_iterator Model::getPointsVecBegin ( ) const

  Gets the iterator for the beginning of vector

- ### getPointsVecEnd( )

  std::vector<Point>::const_iterator Model::getPointsVecEnd ( ) const

  Gets the iterator for the end of vector

- ### isPopulated( )

  bool Model::isPopulated ( ) const

  Returns true if slope and intercept are set, false otherwise.

- ### pushPoint( )

  void Model::pushPoint (const Point & p)

  Push a point into the model, selecting if it should be inserted in the positive of negative parts. Also updates model's energy

  Parameters

  p is the point to be added

- ## fuseModel( )
  void Model::fuseModel (const Model & m)

  Fuses the object with model “m”, combining the points and calculating the new best model and energy

  Parameters

  m is the model that will be fused with object

- ## incrementParallelCount( )
  void Model::incrementParallelCount ( )

  Increments the model’s parallel count by 1

- ## resetParallelCount( )
  void Model::resetParallelCount ( )

  Resets the model’s parallel count to 1

- ## calculateFitness( )
  double Model::calculateFitness ( )

  Calculates model’s fitness using $fitness = \frac{b^2 + 10 * Energy}{n^2 * p}$

- ## clearPoints( )
  void Model::clearPoints ( )

  Remove all points attached to model, resetting energy.

- ## friend operator << ( )
  std::ostream & operator << (std::ostream & out, const Model & m)

  Print model object in terminal.

  Parameters

  out is where to print, normally terminal

  m is the object to be printed