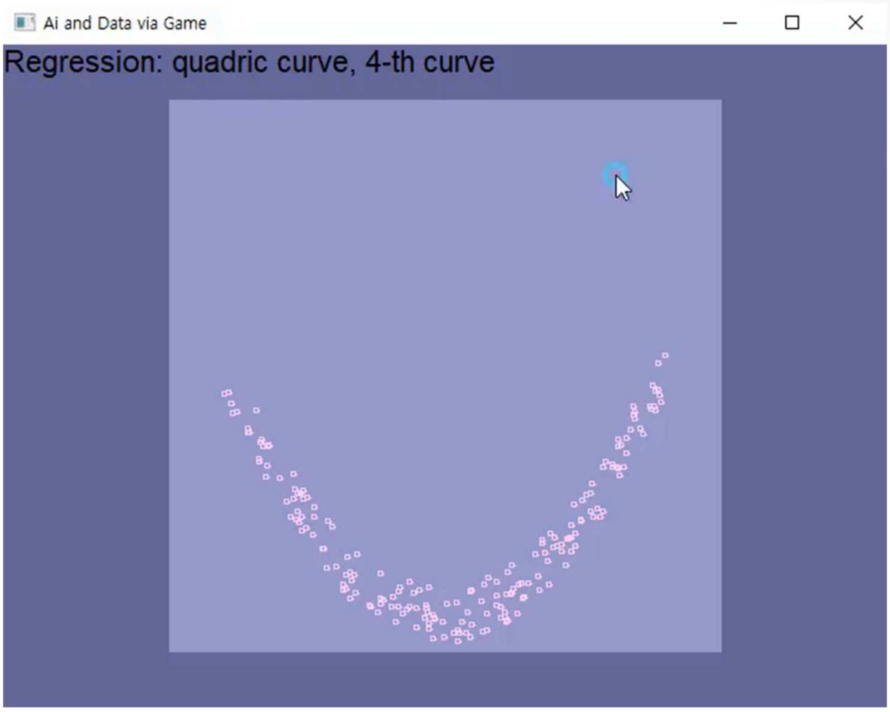
8. Regression



Regression (1)

- Regression
  - Modeling the relationship between a dependent variable and one or more independent variables

$$y = ax + b + \varepsilon$$

$$y = ax_1 + bx_2 + c + \varepsilon$$

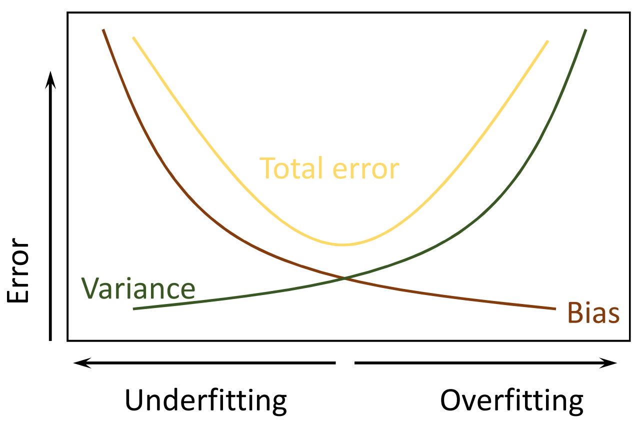
- ε: residual (error)

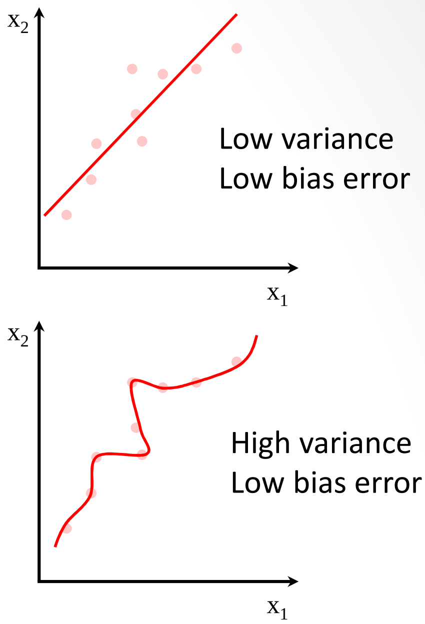
- Linear regression, multiple linear regression, polynomial regression
- Nonlinear regression

$$y = ax_1 + bx_2 + cx_3 + d$$
$$y = ax^3 + bx^2 + cx + d$$

Regression (2)

- Bias-variance trade off
  - Bias error and variance





Regression (3)

- Least squares regression

$$y_1 = ax_1 + b$$
$$y_2 = ax_2 + b$$
$$y_3 = ax_3 + b$$
$$\vdots$$

SSE (Sum of squared errors)

$$\mathbf{Xw} = \hat{\mathbf{y}}$$

$$\mathrm{SSE} = Q = \sum\left(y_i - \hat{y}_i\right)^2 = \sum\left(y_i - ax_i + b\right)^2 = \sum\left(y_i - w_1 x_i + w_0\right)^2$$

$$\frac{\partial Q}{\partial \mathbf{w}} = \frac{\partial}{\partial \mathbf{w}}\left(\mathbf{y} - \mathbf{Xw}\right)^2 = \frac{\partial}{\partial \mathbf{w}}\left(\mathbf{y}^T\mathbf{y} - 2\mathbf{w}^T\mathbf{X}^T\mathbf{y} + \mathbf{w}^T\mathbf{X}^T\mathbf{Xw}\right) = -2\mathbf{X}^T\mathbf{y} + 2\mathbf{X}^T\mathbf{Xw}$$

$$\frac{\partial Q}{\partial \mathbf{w}} \to 0$$

$$-2\mathbf{X}^T\mathbf{y} + 2\mathbf{X}^T\mathbf{Xw}$$

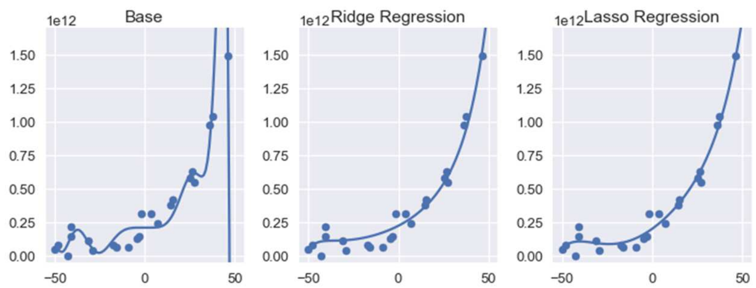
$$\mathbf{X}^T\mathbf{y} = \mathbf{X}^T\mathbf{Xw},\ \mathbf{w} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$$

Regression (4)

- Ridge regression
  - Error + L2 regularization

$$\frac{\partial Q}{\partial \mathbf{w}} = \frac{\partial}{\partial \mathbf{w}}\left(\left(\mathbf{y} - \mathbf{Xw}\right)^2 + \lambda\left\|\mathbf{w}\right\|_2^2\right) \to 0$$

$$\mathbf{w} = \left(\mathbf{X}^T\mathbf{X} + \mathbf{\Gamma}^T\mathbf{\Gamma}\right)^{-1}\mathbf{X}^T\mathbf{y},\ \mathbf{\Gamma} = \alpha\mathbf{I}$$



https://www.textbook.ds100.org/ch/16/reg_lasso.html

$$+\lambda\sum_{i=0}^{n}\left|w_i\right|$$

$$\lambda\frac{\partial}{\partial \mathbf{w}}\left(\left\|\mathbf{w}\right\|_1\right) = \lambda\sum\frac{\partial}{\partial w_i}\left(\left|w_i\right|\right)$$

- Lasso regression
  - Least absolute shrinkage and selection operator
  - Error + L1 regularization

$$\lambda\frac{\partial}{\partial w_i}\left(\left|w_i\right|\right) = \begin{cases} -\lambda & w_i < 0 \\ [-\lambda, \lambda] & w_i = 0 \\ \lambda & w_i > 0 \end{cases}$$

Least Squares (1)

```
bool LeastSquared(double **X, double *w, double *y, int nRow, int nCol,
  bool bRidge, double alpha) {
  double **Xt = dmatrix(nCol, nRow);
  double **XtX = dmatrix(nCol, nCol);
  double **InverseXtX = dmatrix(nCol, nCol);
  double **PseudoInverseX = dmatrix(nCol, nRow);

  for(int r = 0 ; r < nCol ; ++r)
    for(int c = 0 ; c < nRow ; ++c)
      Xt[r][c] = X[c][r];

  for(int r = 0 ; r < nCol ; ++r)
    for(int c = 0 ; c < nCol ; ++c) {
      XtX[r][c] = 0;
      for(int k = 0 ; k < nRow ; ++k)
        XtX[r][c] += Xt[r][k] * X[k][c];

      if(bRidge)
        if(r == c) XtX[r][c] += alpha*alpha;
    }
```

$$\mathbf{w} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$$

Least Squares (2)

```
  if(InverseMatrix(XtX, InverseXtX, nCol)) {
    for(int r = 0 ; r < nCol ; ++r)
      for(int c = 0 ; c <  nRow ; ++c) {
        PseudoInverseX[r][c] = 0;
        for(int k = 0 ; k < nCol ; ++k)
          PseudoInverseX[r][c] += InverseXtX[r][k] *Xt[k][c];
      }
    for(int r = 0 ; r < nCol ; ++r) {
      w[r] = 0;
      for(int k = 0 ; k < nRow ; ++k)
        w[r] += PseudoInverseX[r][k] * y[k];
    }
  }
  else {
    free_dmatrix(Xt, nCol, nRow);
    free_dmatrix(XtX, nCol, nCol);
    free_dmatrix(InverseXtX, nCol, nCol);
    free_dmatrix(PseudoInverseX, nCol, nRow);

    return false;
  }
```

$$\mathbf{w} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$$

Least Squares (3)

```
  free_dmatrix(Xt, nCol, nRow);
  free_dmatrix(XtX, nCol, nCol);
  free_dmatrix(InverseXtX, nCol, nCol);
  free_dmatrix(PseudoInverseX, nCol, nRow);
  return true;
}
```


Main.cpp (1)

```
…

class CLsmLayer : public CKhuGleLayer {
public:
  std::vector<CKhuGleSprite *> m_Point;
  bool m_bQuadricCurve;
  int m_nPointCnt;
  double **m_X, *m_y, *m_w;

  CLsmLayer(int nW, int nH, KgColor24 bgColor, CKgPoint ptPos = CKgPoint(0, 0),
    int nPointCnt = 100) : CKhuGleLayer(nW, nH, bgColor, ptPos) {
    m_X = nullptr;
    m_y = nullptr;
    m_w = nullptr;

    m_bQuadricCurve = true;

    GenerateData(nPointCnt, false);
  }
  virtual ~CLsmLayer() {
    if(m_X) free_dmatrix(m_X, m_nPointCnt, 3);
    if(m_y) delete [] m_y;
    if(m_w) delete [] m_w;
  }
  void GenerateData(int nCnt, bool bExtremeNoise);
};
```

Main.cpp (2)

```
void CLsmLayer::GenerateData(int nCnt, bool bExtremeNoise) {
  if(m_X) free_dmatrix(m_X, m_nPointCnt, 3);
  if(m_y) delete [] m_y;
  if(m_w) delete [] m_w;

  m_nPointCnt = nCnt;
  m_X = dmatrix(m_nPointCnt, 3);
  m_y = new double[m_nPointCnt];
  m_w = new double[3];

  unsigned int seed
    = (unsigned  int)std::chrono::system_clock
      ::now().time_since_epoch().count();
  std::default_random_engine generator(seed);
```


Main.cpp (3)

```
  std::uniform_real_distribution<double> uniform_dist1(0.005, 0.01);
  std::uniform_real_distribution<double> uniform_dist2(m_nW*0.4, m_nW*0.6);
  std::uniform_real_distribution<double> uniform_dist3(m_nH*0.9, m_nH*0.95);
  std::uniform_real_distribution<double> uniform_dist4(m_nW*0.1, m_nW*0.9);
  std::uniform_real_distribution<double> uniform_dist5(0, m_nW*0.1);
  double a = -uniform_dist1(generator);
  double x0 = uniform_dist2(generator);
  double y0 = uniform_dist3(generator);
  double ExtremeNoisePos = uniform_dist4(generator);
```

Main.cpp (4)

```
  for(auto &Child : m_Children)
    delete Child;
  m_Children.clear();
  m_Point.clear();

  double x, y, noise;
  double m = (rand()%2?1:-1)*a*100;
  for(int i = 0 ; i < m_nPointCnt ; ++i) {
    noise = uniform_dist5(generator)-m_nW*0.05;
    x = uniform_dist4(generator);

    if(m_bQuadricCurve)    y = a*(x-x0)*(x-x0) + y0 + noise;
    else    y = m*(x-x0) + y0 + noise;

    if(bExtremeNoise) {
      if(x > ExtremeNoisePos-m_nW*0.05 && x < ExtremeNoisePos+m_nW*0.05) {
        if(m_bQuadricCurve)
          y = a*(x-x0)*(x-x0) + y0 + (noise-m_nW*0.05)*3;
        else
          y = m*(x-x0) + y0 + (noise-m_nW*0.05)*3;
    }
  }
```


Main.cpp (5)

```
  m_X[i][0] = x*x;
  m_X[i][1] = x;
  m_X[i][2] = 1;

  m_y[i] = y;

  CKhuGleSprite *Point = new CKhuGleSprite(GP_STYPE_ELLIPSE, GP_CTYPE_DYNAMIC,
  CKgLine(CKgPoint((int)x-2, (int)y-2), CKgPoint((int)x+2, (int)y+2)),
  KG_COLOR_24_RGB(255, 200, 255), false, 30);

  m_Point.push_back(Point);
  AddChild(Point);
  }
  SetBackgroundImage(m_nW, m_nH, m_bgColor);
}

class CRegression : public CKhuGleWin {
public:
  CLsmLayer *m_pLsmLayer;

  CRegression(int nW, int nH);
  void Update();
};
```

Main.cpp (6)

```
CRegression::CRegression(int nW, int nH) : CKhuGleWin(nW, nH) {
  m_pScene = new CKhuGleScene(640, 480, KG_COLOR_24_RGB(100, 100, 150));
  m_pLsmLayer = new CLsmLayer(400, 400, KG_COLOR_24_RGB(150, 150, 200),
    CKgPoint(120, 40), 200);
  m_pScene->AddChild(m_pLsmLayer);
}
void CRegression::Update() {
  if(m_bKeyPressed['Q']) {
    m_pLsmLayer->m_bQuadricCurve = !m_pLsmLayer->m_bQuadricCurve;
    m_pLsmLayer->GenerateData(200, false);
    m_bKeyPressed['Q'] = false;
  }
```


Main.cpp (7)

```
 if(m_bKeyPressed['S']) {
    LeastSquared(m_pLsmLayer->m_X, m_pLsmLayer->m_w, m_pLsmLayer->m_y,
      m_pLsmLayer->m_nPointCnt, 3, false, 0);

    int y0;
    for(int x = 0 ; x < m_pLsmLayer->m_nW ; ++x) {
      int y = (int)(m_pLsmLayer->m_w[0]*x*x + m_pLsmLayer->m_w[1]*x
                    + m_pLsmLayer->m_w[2]);
      if(x > 0) {
        CKhuGleSprite::DrawLine(m_pLsmLayer->m_ImageBgR,
          m_pLsmLayer->m_ImageBgG, m_pLsmLayer->m_ImageBgB,
          m_pLsmLayer->m_nW, m_pLsmLayer->m_nH, x-1, y0,
          x, y, KG_COLOR_24_RGB(255, 0, 0));
      }
      y0 = y;
    }
```

Main.cpp (8)

```
    LeastSquared(m_pLsmLayer->m_X, m_pLsmLayer->m_w, m_pLsmLayer->m_y,
      m_pLsmLayer->m_nPointCnt, 3, true, 0.9);

    for(int x = 0 ; x < m_pLsmLayer->m_nW ; ++x) {
      int y = (int)(m_pLsmLayer->m_w[0]*x*x + m_pLsmLayer->m_w[1]*x
                    + m_pLsmLayer->m_w[2]);
      if(x > 0) {
        CKhuGleSprite::DrawLine(m_pLsmLayer->m_ImageBgR,
          m_pLsmLayer->m_ImageBgG, m_pLsmLayer->m_ImageBgB,
          m_pLsmLayer->m_nW, m_pLsmLayer->m_nH, x-1, y0, x, y,
          KG_COLOR_24_RGB(255, 255, 0));
      }
    y0 = y;
    }
    m_bKeyPressed['S'] = false;
  }
```

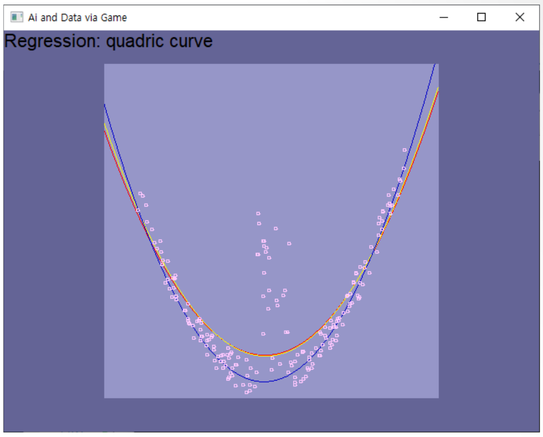

Main.cpp (9)

```
  if(m_bKeyPressed['N'] || m_bKeyPressed['M']) {
    if(m_bKeyPressed['M'])   m_pLsmLayer->GenerateData(200, true);
    else   m_pLsmLayer->GenerateData(200, false);

    m_bKeyPressed['N'] = false;
    m_bKeyPressed['M'] = false;
  }
  m_pScene->Render();

  if(m_pLsmLayer->m_bQuadricCurve)
    DrawSceneTextPos("Regression: quadric curve", CKgPoint(0, 0));
  else
    DrawSceneTextPos("Regression: line", CKgPoint(0, 0));
  CKhuGleWin::Update();
}
int main() {
  CRegression *pRegression = new CRegression(640, 480);
  KhuGleWinInit(pRegression);
  return 0;
}
```

Practice VI

- RANSAC (Random sample consensus)
  - Iterative parameter estimation method

  - **W** ← Ø
    C_M ←0
    while iteration **do**
    randomly subset selection
    estimate parameter ($\mathbf{W}_i$)
    inlier count (C_i)
    if $C_i > C_M$ **then**
    $C_M \leftarrow C_i$
    $\mathbf{W} \leftarrow \mathbf{W}_i$
    end if
    end while

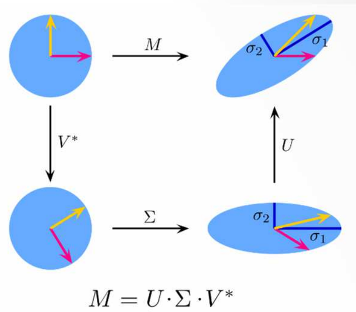


Advanced Courses (1)

- Singular value decomposition (SVD)

  $$\mathbf{M} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$$

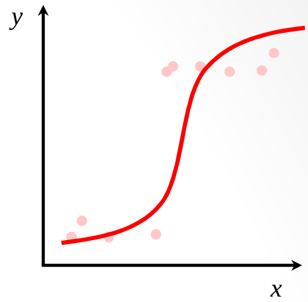
  - **U** and **V** are singular vectors, orthonormal and unitary matrices
  - **Σ** is a diagonal matrix having singular values
  - Applications
    - Pseudo inverse
    - Truncated SVD
      - Regularization
      - Dimensionality reduction



https://upload.wikimedia.org/wikipedia/commons/thumb/b/bb/Singular-Value-Decomposition.svg/220px-Singular-Value-Decomposition.svg.png

Advanced Courses (2)

- Logistic regression
  - Classification using a logistic function



$$P(y=1)=\frac{1}{1+e^{-(b+\mathbf{wX})}}$$

$$\frac{1}{P(y=1)}-1=e^{-(b+\mathbf{wX})},\quad \frac{1-P(y=1)}{P(y=1)}=\frac{1}{e^{(b+\mathbf{wX})}}$$

$$\log\left(\frac{P(y=1)}{1-P(y=1)}\right)=\log\left(\frac{P(y=1)}{P(y=0)}\right)=b+\mathbf{wX}$$