1 Initial Value Problem

1.1 General solvability theory

In this section we study the numerical solution for the following initial value problem :

$$(IVP) \qquad \begin{cases} Y'(t) = f(t, Y(t)), & t \geq t_0 \\ Y(t_0) = Y_0. \end{cases}$$

Theorem 1.1. *If $f(t,z)$ and $\partial f(t,z)/\partial z$ be continuous functions of t and z at all points (t,z) in some neighborhood of the initial point (t_0, Y_0). Then there is a unique function $Y(t)$ defined on some interval $[t_0 - \alpha, t_0 + \alpha]$, satisfying*

$$\begin{cases} Y'(t) = f(t, Y(t)), & t_0 - \alpha \leq t \leq t_0 + \alpha \\ Y(t_0) = Y_0. \end{cases}$$

Note.

1. The number α depends on the initial value problem(IVP).
2. For some equations, e.g., $f(t,z) = \lambda z + b(t)$ where $b(t)$ is continuous, solutions exist for any t, i.e., we can take α to be ∞.
3. For many nonlinear equations, solutions can exist only in bounded intervals.

1.2 Stability

We will generally assume that the solution $Y(t)$ is being sought on a given finite interval $t_0 \le t \le T$.

When small changes in the initial value Y_0 leads to small changes in the solution $Y(t)$, the IVP is said to be stable.

That is, for the solution $Y_\eta(t)$ of the perturbed problem

$$Y'_\varepsilon(t) = f(t, Y_\varepsilon(t)), \quad t_0 \le t \le T, \quad Y_\varepsilon(t_0) = Y_0 + \varepsilon,$$

we have

$$\max_{x_0 \le t \le T} |Y_\varepsilon(t) - Y(t)| \le c\,\varepsilon, \quad \text{some} \;\; c > 0.$$

Example (Stable)**.**

$$Y'(t) = -Y(t) + 1, \quad 0 \le t \le b, \quad Y(0) = 1, \qquad \text{Solution}: \;\; Y(t) \equiv 1,$$

$$Y'_\varepsilon(t) = -Y_\varepsilon(t) + 1, \quad 0 \le t \le b, \quad Y_\varepsilon(0) = 1 + \varepsilon, \qquad \text{Solution}: \;\; Y_\varepsilon(t) \equiv 1 + \varepsilon e^{-t}.$$

$$\Longrightarrow \quad |Y_\varepsilon(t) - Y(t)| = |-\varepsilon e^{-t}| \le |\varepsilon|, \quad t \ge 0.$$

Example (Ill-conditioned when $\lambda b > 0$ is large.)**.**

$$Y'(t) = \lambda[Y(t) - 1], \quad 0 \le t \le b, \quad Y(0) = 1, \qquad \text{Solution}: \;\; Y(t) \equiv 1,$$

$$Y'_\varepsilon(t) = \lambda[Y_\varepsilon(t) - 1], \quad 0 \le t \le b, \quad Y_\varepsilon(0) = 1 + \varepsilon, \qquad \text{Solution}: \;\; Y_\varepsilon(t) \equiv 1 + \varepsilon e^{\lambda t}.$$

$$\Longrightarrow \quad \max_{0 \le t \le b} |Y_\varepsilon(t) - Y(t)| = |\varepsilon|\, e^{\lambda b}, \quad \text{the change in } Y(t) \text{ is quite significant at } t = b.$$

1.3 Direction Fields

Direction fields are a useful tool to understand the behavior of solutions of a differential equation.

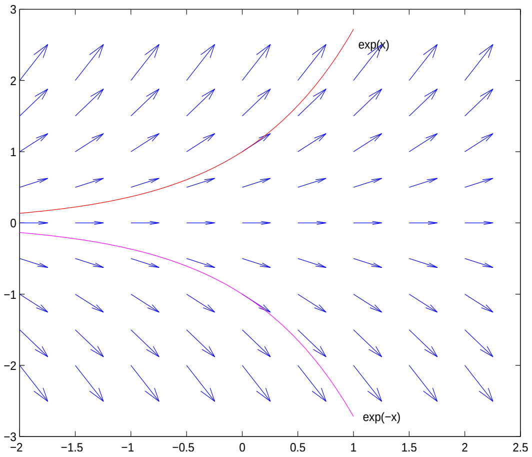
In the graph of a solution of $y' = f(x,y)$, the slope is $f(x,y)$ regardless of the initial value.

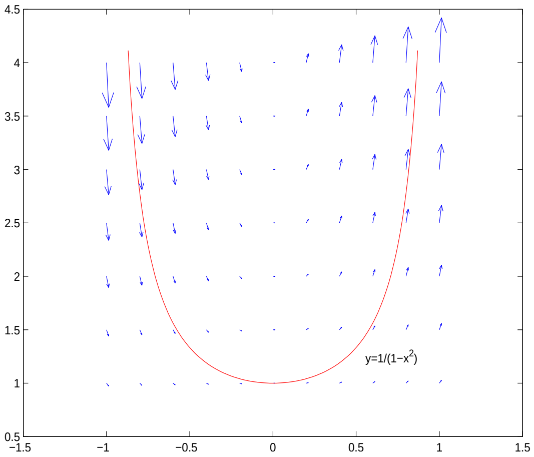
Example. General solution is $y = c\,e^x$ for the D.E. $y' = y$.

Example. General solution is $y = c/(1-x^2)$ for the D.E. $y' = 2xy^2$.

```
[x,y] = meshgrid(-2:0.5:2, -2:0.5:2);
dx = ones(9);  % one matrix
dy = y; quiver(x,y,dx,dy);
hold on
x = -2:0.01:1;
y1 = exp(x);  y2 = -exp(x);
plot(x,y1,'r',x,y2,'m')
```

```
[x,y] = meshgrid(-1:0.2:1, 1:0.5:4);
dx = ones(7,11);  % one matrix
dy = 2*x.*y.^2; quiver(x,y,dx,dy);
hold on
x = -0.87:0.01:0.87;
y = 1./(1-x.^2);
plot(x,y,'r')
```





1.4 Euler's Method

The simplest, but not efficient numerical method.

The approximate solution will be denoted by $y(t)$ and denote by $y_n = y(t_n)$.

Note that the derivative approximation :

$$Y'(t) \approx \frac{Y(t+h) - Y(t)}{h}.$$

Take the nodes as

$$t_n = t_0 + n * h, \quad n = 0, 1, 2, \cdots, N, \quad \text{with} \;\; h = (T - t_0)/N.$$

Applying the derivative approximation to the IVP :

$$Y'(t_n) = f(t_n, Y(t_n)) \quad \text{at} \;\; t = t_n,$$

we have

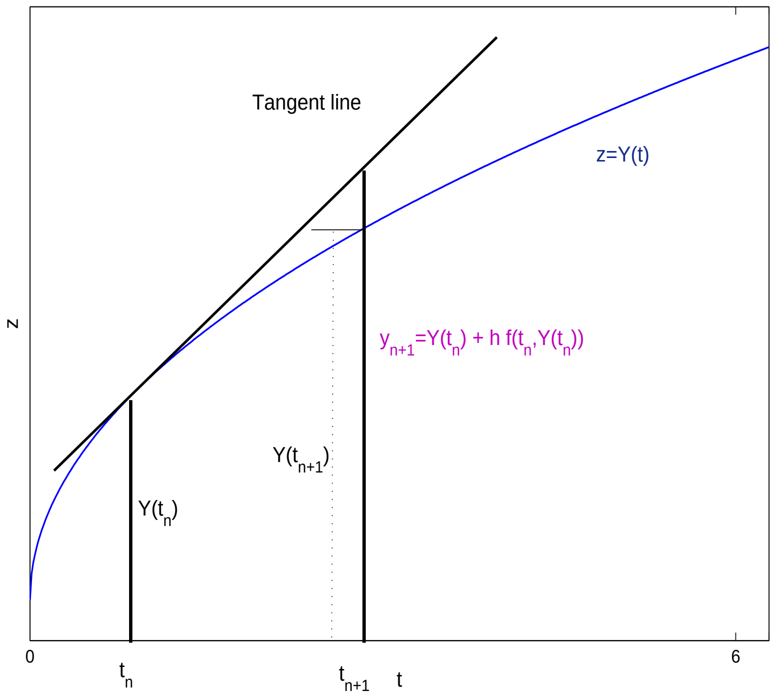
$$\frac{Y(t_{n+1}) - Y(t_n)}{h} \approx f(t_n, Y(t_n)).$$

Euler's method is defined as, with $y_0 = Y_0$

$$\boxed{y_{n+1} = y_n + h\, f(t_n, y_n), \quad n = 0, 1, 2, \cdots, N-1.}$$

Example. Solve $Y'(t) = \dfrac{Y(t) + t^2 - 2}{t+1}, \quad Y(0) = 2$

whose true solution is $Y(t) = t^2 + 2t + 2 - 2(t+1)\log(t+1)$.



1.5 Taylor Methods

Euler's method uses a linear Taylor polynomial approximation :

$$Y(t_{n+1}) \approx Y(t_n) + h\,Y'(t_n) = Y(t_n) + h\,f(t_n, Y(t_n)).$$

Note that the following Chain rule : with $z'(t) = f\big(t, z(t)\big)$

$$z''(t) = \frac{df(t,z(t))}{dt} = \frac{\partial f}{\partial t}\big(t,z(t)\big) + \frac{\partial f}{\partial z}\big(t,z(t)\big)\,z'(t) = f_t\big(t,z(t)\big) + f_z\big(t,z(t)\big)\,f\big(t,z(t)\big).$$

Using a quadratic Taylor polynomial approximation :

$$Y(t_{n+1}) \approx Y(t_n) + h\,Y'(t_n) + \frac{h^2}{2}Y''(t_n),$$

we have the second order Taylor approximation :

$$Y(t_{n+1}) \approx Y(t_n) + h\,f(t_n, Y(t_n)) + \frac{h^2}{2}\left[f_t\big(t_n, Y(t_n)\big) + f_z\big(t_n, Y(t_n)\big)\,f\big(t_n, Y(t_n)\big)\right].$$

The second-order($p = 2$) Taylor method for IVP is given by

$$\boxed{y_{n+1} = y_n + h\,f(t_n, y_n) + \frac{h^2}{2}\left[f_t\big(t_n, y_n\big) + f_z\big(t_n, y_n\big)\,f\big(t_n, y_n\big)\right], \quad \text{Error : } |Y(t_n) - y(t_n)| = O\big(h^{p+1}\big).}$$

Example. Solve

$$Y'(t) = -Y(t) + 2\cos t, \quad Y(0) = 1 \quad \text{whose true solution is } Y(t) = \sin t + \cos t.$$

$$\text{Since } Y''(t) = -Y'(t) - 2\sin t = Y(t) - 2\cos t - 2\sin t,$$

$$\text{Taylor Scheme : } y_{n+1} = y_n + h[-y_n + 2\cos t_n] + \frac{h^2}{2}\left[y_n - 2\cos t_n - 2\sin t_n\right], \quad n \geq 0.$$

1.6 Runge-Kutta Methods

The Taylor method needs to calculate the higher-order derivatives.

The Runge-Kutta methods evaluate $f(t,z)$ at more points,

while attempting to equal the accuracy of the Taylor approximation.

RK methods are among the most popular methods for solving IVP.

The Runge-Kutta methods of order 2 : RK2

The general form

$$y_{n+1} = y_n + h\,F(t_n, y_n; h), \quad n \geq 0, \quad y_0 = Y_0$$

where $F(t_n, y_n; h)$ can be thought of as some kind of ”average slope” of the solution on the interval $[t_n, t_{n+1}]$.

Choose

$$F(t, y; h) = \gamma_1\, f(t, y) + \gamma_2\, f\big(t + \alpha h, y + \beta h f(t, y)\big).$$

Determine the constants $\alpha, \beta, \gamma_1, \gamma_2$ so that the following truncation error will be $O(h^3)$,

just as with the Taylor method of order 2 :

$$T_{n+1} = Y(t_{n+1}) - [Y(t_n) + h\,F(t_n, Y(t_n); h)].$$

Note that the first-order Taylor expansion for two-variable function $f\big(t, y\big)$:

$$f\big(t + A, y + B\big) = f\big(t, y\big) + A\,f_t(t, y) + B\,f_z(t, y) + O(A^2 + B^2).$$

Hence, we have

$$f\big(t + \alpha h, y + \beta h f(t, y)\big) = f\big(t, y\big) + \alpha\,h\,f_t(t, y) + \beta\,h\,f_z\big(t, y\big)\,f(t, y) + O(h^2).$$

Using $Y'' = f_t + f_z\, f$ yields

$$\begin{aligned} Y(t+h) &= Y + h\,Y' + \frac{h^2}{2} Y'' + O(h^3) \\ &= Y + h\,f + \frac{h^2}{2}\big(f_t + f_z\, f\big) + O(h^3). \end{aligned}$$

Then

$$\begin{aligned} &Y(t+h) - \big[Y(t) + h\,F(t, Y(t); h)\big] \\ &\quad = \Big[Y + h\,f + \frac{h^2}{2}(f_t + f_z\, f)\Big] - \Big[Y + h\,\gamma_1\, f + h\,\gamma_2\, f\big(t + \alpha h, Y + \beta h f(t, Y)\big)\Big] + O(h^3) \\ &\quad = \Big[Y + h\,f + \frac{h^2}{2}(f_t + f_z\, f)\Big] - \Big[Y + h\,\gamma_1\, f + h\,\gamma_2\,\big(f + \alpha\, h\, f_t + \beta\, h\, f_z\, f\big)\Big] + O(h^3) \\ &\quad = h\,(1 - \gamma_1 - \gamma_2)\, f + \frac{h^2}{2}\Big[(1 - 2\alpha\gamma_2) f_t + (1 - 2\beta\gamma_2) f_z\, f\Big] + O(h^3). \end{aligned}$$

So the coefficients must satisfy the system

$$1 - \gamma_1 - \gamma_2 = 0, \quad 1 - 2\alpha\gamma_2 = 0, \quad 1 - 2\beta\gamma_2 = 0.$$

Therefore,

$$\gamma_2 \neq 0, \quad \gamma_1 = 1 - \gamma_2, \quad \alpha = \beta = \frac{1}{2\gamma_2}.$$

The three favorite choices are $\gamma_2 = \frac{1}{2}, \frac{3}{4}, 1$.

With $\gamma_2 = 1/2$, we have the following RK2 method :

$$\boxed{y_{n+1} = y_n + \frac{h}{2}\Big[f(t_n, y_n) + f\big(t_n + h, y_n + h f(t_n, y_n)\big)\Big].}$$

```
----------------------------------------

 Runge-Kutta Method of order 2

----------------------------------------

   v1 = f( t(n), y(n) );

   v2 = f( t(n)+h, y(n)+h*v1 );

 y(n+1) = y(n) + (h/2)*( v1 + v2 );

-----------------------------------------
```

The Runge-Kutta methods of order 4 : RK4

The truncation error in this method is $O(h^5)$:

$$\begin{aligned}
v_1 &= f(t_n, y_n) \\
v_2 &= f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}v_1\right) \\
v_3 &= f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}v_2\right) \\
v_4 &= f(t_n + h, y_n = h\, v_3) \\
y_{n+1} &= y_n + \frac{h}{6}\big[v_1 + 2v_2 + 2v_3 + v_4\big].
\end{aligned}$$

Example. Using RK2 and RK4 with $h = 0.1$ and 0.05, solve

$$Y'(t) = \frac{1}{4}Y(t)\left(1 - \frac{1}{20}Y(t)\right) \quad 0 \le t \le 20, \quad Y(0) = 1 \quad \text{whose true solution is} \quad Y(t) = \frac{20}{1 + 19\, e^{-x/4}}.$$

```
--------------------------------------------------
   h     t(n)      Y(n)      y(n)       |Y(n)-y(n)|
--------------------------------------------------
  0.1    2.0
         4.0
          .
--------------------------------------------------
```

1.7 The 2nd-order initial value problem

Consider

$$(IVP) \qquad \begin{cases} y''(t) + a(t)y'(t) = f(t, y(t)), & t \geq t_0 \\ y(t_0) = \alpha, \quad y'(t_0) = \beta. \end{cases}$$

Convert to an equivalent system of 1st-order IVP

Set $\quad y_1(t) = y(t), \quad y_2(t) = y'(t)$

$$(IVP) \qquad \begin{cases} y_1'(t) = y_2(t), & y_1(t_0) = \alpha \\ y_2'(t) = f(t, y_1(t)) - a(t)y_2(t), & y_2(t_0) = \beta \end{cases} \quad \Leftrightarrow \quad Y'(t) = F(t, Y(t)), \quad Y(t_0) = Y_0$$

```
--------------------------------------------------------------------------
 Runge-Kutta Method of order 2 : V1, V2, Y(n) are all vector values
--------------------------------------------------------------------------
  V1 = f( t(n), Y(n) );  V2 = f( t(n)+h, Y(n)+h*V1 );
  Y(n+1) = Y(n) + (h/2)*( V1 + V2 );
```

Example. Using RK2 and RK4 with $h = 0.1$ and 0.05, solve

$$y''(t) + 11y'(t) + 10y(t) = 10, \quad y(0) = 0,\ y'(0) = 0 \quad \text{whose true solution is} \quad y(t) = 1 - \frac{10}{9}\,e^{-t} + \frac{1}{9}\,e^{-10t}.$$

2 Introduction to Finite Difference Methods

2.1 FDM for 1D Problems

Let $h = 1/N$ and set

$$t_i = i\,h, \quad I_i = [t_{i-1}, t_i],$$

$$0 = t_0 < t_1 < t_2 < \cdots < t_{N-1} < t_N = 1.$$

Denote by the value of a function $\psi_i = \psi(t_i)$. Also denote by $u'_i = u'(t_i)$ and $u''_i = u''(t_i)$.

Define a numerical derivative of $u(x)$ with stepsize h:

$$D_h u(t) := \frac{u(t+h) - u(t)}{h} \approx u'(t).$$

From Taylor expansion,

$$u(t_i + h) = u(t_i) + u'(t_i)h + \frac{u''(t_i)}{2}h^2 + \frac{u^{(3)}(t_i)}{3!}h^3 + \frac{u^{(4)}(\xi_i)}{4!}h^4$$
$$u(t_i - h) = u(t_i) - u'(t_i)h + \frac{u''(t_i)}{2}h^2 - \frac{u^{(3)}(t_i)}{3!}h^3 + \frac{u^{(4)}(\xi_i)}{4!}h^4.$$

or

$$
\begin{aligned}
u'(t_i) &= \frac{u(t_i+h)-u(t_i)}{h} - \frac{u''(t_i)}{2}h - \frac{u^{(3)}(t_i)}{3!}h^2 + \frac{u^{(4)}(\xi_i)}{4!}h^3 \\
u'(t_i) &= \frac{u(t_i)-u(t_i-h)}{h} + \frac{u''(t_i)}{2}h - \frac{u^{(3)}(t_i)}{3!}h^2 + \frac{u^{(4)}(\xi_i)}{4!}h^3 \\
u'(t_i) &= \frac{u(t_i+h)-u(t_i-h)}{2h} - \frac{u^{(3)}(t_i)}{3!}h^2 + \frac{u^{(4)}(\xi_i)}{4!}h^3 \\
u''(t_i) &= \frac{u(t_i-h)-2u(t_i)+u(t_i-h)}{h^2} - \frac{u^{(3)}(t_i)}{3!}h^2 + \frac{u^{(4)}(\xi_i)}{4!}h^3.
\end{aligned}
$$

Now, define the following finite differences:

1. The forward finite difference of the first order derivative:

$$u'(t_i) = \frac{u_{i+1} - u_i}{h} + O(h)$$

2. The backward finite difference of the first order derivative:

$$u'(t_i) = \frac{u_i - u_{i-1}}{h} + O(h)$$

3. The central finite difference of the first order derivative:

$$u'(t_i) = \frac{u_{i+1} - u_{i-1}}{2h} + O(h^2)$$

4. The central finite difference of the second order derivative:

$$u''(t_i) = \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} + O(h^2).$$

Here, one may easily check the error bounds using the Taylor series expansion with the sufficient smooth function u.

Note that the discretization error bound is of the second order if we use the central finite difference formula and one may use more points to get better error bounds.

Consider the following 1-D problem, an initial value problem:

$$\begin{cases} u' + cu = f \qquad \text{in } (0,1), \\ u(0) = \alpha, \end{cases} \tag{1D}$$

Then we have the following approximation scheme using the backward finite difference:

$$\text{With} \quad u_0 = \alpha,$$

$$\frac{u_i - u_{i-1}}{h} + c_i u_i = f_i, \quad i = 1, 2, \cdots, N,$$

or

$$u_i = \left(\frac{1}{1 + c_i\, h}\right) u_{i-1} + \left(\frac{h}{1 + c_i\, h}\right) f_i, \quad i = 1, 2, \cdots, N.$$

Also let us consider the following 1-D problem, two-point boundary value problem:

$$\begin{cases} -u'' + bu' + cu = f \quad \text{in } (0,1), \\ u(0) = \alpha, \quad u(1) = \beta. \end{cases} \tag{1D}$$

Then we have the following approximation scheme using the central finite differences:

$$\text{With} \quad u_0 = \alpha \quad \text{and} \quad u_N = \beta,$$

$$-\frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} + b_i \frac{u_{i+1} - u_{i-1}}{2h} + c_i u_i = f_i, \quad i = 1, 2, \cdots, N-1$$

or

$$\left(-1 - \frac{b_i\, h}{2}\right) u_{i-1} + (c_i\, h^2 + 2) u_i + \left(-1 + \frac{b_i\, h}{2}\right) u_{i+1} = f_i\, h^2, \quad i = 1, 2, \cdots, N-1.$$

Then, we have the following tridiagonal system when $b = c = 0$:

$$A\, \hat{u} = \hat{f},$$

where

$$A = \text{diag}(-1, 2, -1) := \frac{1}{h^2} * \begin{bmatrix} 2 & -1 & 0 & 0 & \cdots & 0 \\ -1 & 2 & -1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & -1 & 2 & -1 \\ 0 & \cdots & \cdots & 0 & -1 & 2 \end{bmatrix}, \quad \hat{u} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{N-2} \\ u_{N-1} \end{bmatrix} \quad \text{and} \quad \hat{f} = \begin{bmatrix} f_1 + \alpha/h^2 \\ f_2 \\ f_3 \\ \vdots \\ f_{N-2} \\ f_{N-1} + \beta/h^2 \end{bmatrix}.$$

2.2 FDM for 2D Problems

Let $U = \big(u_{j,i}\big)$ and $F = \big(f_{j,i}\big)$ be matrices consisting of values of function u and f on the grid points (x_j, y_i) such as

$$U = \begin{bmatrix} u_{1,1} & u_{2,1} & \cdots & u_{N,1} \\ u_{1,2} & u_{2,2} & \cdots & u_{N,2} \\ \vdots & \vdots & \ddots & \vdots \\ u_{1,N} & u_{2,N} & \cdots & u_{N,N} \end{bmatrix} \quad \text{and} \quad F = \begin{bmatrix} f_{1,1} & f_{2,1} & \cdots & f_{N,1} \\ f_{1,2} & f_{2,2} & \cdots & f_{N,2} \\ \vdots & \vdots & \ddots & \vdots \\ f_{1,N} & f_{2,N} & \cdots & f_{N,N} \end{bmatrix}$$

To write a linear system in the standard form $A\mathbf{x} = \mathbf{b}$, we need to order the unknown u_{ij} in some way.

For $U = \big(u_{j,i}\big) \in \mathbb{R}^{n,n}$, we define a vector

$$vecU = \big(u_{1,1}, \cdots, u_{1,n}, u_{2,1}, \cdots, v_{2,n}, \cdots, u_{n,1}, \cdots, u_{n,n}\big)^T \in \mathbb{R}^{n^2}$$

by stacking the columns of U on top of each other.

Note that

$$u_{i,j} = vecU\big((i-1)n + j\big).$$

We also need to define a Kronecker product.

For $A \in \mathbb{R}^{p,q}$ and $B \in \mathbb{R}^{r,s}$, define the (right) Kronecker (tensor) product

$$C = A \otimes B := \begin{bmatrix} a_{1,1}B & a_{1,2}B & \cdots & a_{1,q}B \\ a_{2,1}B & a_{2,2}B & \cdots & a_{2,q}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{p,1}B & a_{p,2}B & \cdots & a_{p,q}B \end{bmatrix}.$$

If the system has the following five points stencil

$$\begin{bmatrix} 0 & \beta_3 & 0 \\ \alpha_1 & \alpha_2+\beta_2 & \alpha_3 \\ 0 & \beta_1 & 0 \end{bmatrix},$$

then, without touching the boundary, it can be written by

$$\beta_1\,\hat{u}_{\ell-1}+\beta_2\,\hat{u}_\ell+\beta_3\,\hat{u}_{\ell+1}+\alpha_1\,\hat{u}_{\ell-N}+\alpha_2\,\hat{u}_\ell+\alpha_3\,\hat{u}_{\ell+N}=\hat{f}_\ell$$

where $\hat{f}_\ell = f_{i,j}$ and $\hat{u}_\ell = u_{i,j}$.

Let $J_1 = \text{diag}(\alpha_1,\alpha_2,\alpha_3)$ and $J_2 = \text{diag}(\beta_1,\beta_2,\beta_3)$:

$$J_1 = \text{diag}(\alpha_1,\alpha_2,\alpha_3) := \begin{bmatrix} \alpha_2 & \alpha_3 & 0 & 0 & \cdots & 0 \\ \alpha_1 & \alpha_2 & \alpha_3 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \alpha_1 & \alpha_2 & \alpha_3 \\ 0 & \cdots & \cdots & 0 & \alpha_1 & \alpha_2 \end{bmatrix} \quad \text{and} \quad J_2 = \text{diag}(\beta_1,\beta_2,\beta_3) := \begin{bmatrix} \beta_2 & \beta_3 & 0 & 0 & \cdots & 0 \\ \beta_1 & \beta_2 & \beta_3 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \beta_1 & \beta_2 & \beta_3 \\ 0 & \cdots & \cdots & 0 & \beta_1 & \beta_2 \end{bmatrix}.$$

Then the matrix is given by the block form

$$A = \begin{bmatrix} \alpha_2\, I & \alpha_3\, I & 0 & 0 & \cdots & 0 \\ \alpha_1\, I & \alpha_2\, I & \alpha_3\, I & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \alpha_1\, I & \alpha_2\, I & \alpha_3\, I \\ 0 & \cdots & \cdots & 0 & \alpha_1\, I & \alpha_2\, I \end{bmatrix} + \begin{bmatrix} J_2 & 0 & 0 & 0 & \cdots & 0 \\ 0 & J_2 & 0 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 0 & J_2 & 0 \\ 0 & \cdots & \cdots & 0 & 0 & J_2 \end{bmatrix}.$$

Furthermore, the matrix A is also given by the Kronecker Products:

$$A = J_1 \otimes I + I \otimes J_2.$$

```
-------------------------------------------------------------------------------

In Matlab
  >> J1 = alp1*diag(ones(N-1,1),-1) + alp2*diag(ones(N,1)) + alp3*diag(ones(N-1,1),1);
  >> J2 = bet1*diag(ones(N-1,1),-1) + bet2*diag(ones(N,1)) + bet3*diag(ones(N-1,1),1);
  >> A = kron(J1,eye(N)) + kron(eye(N),J2);
-------------------------------------------------------------------------------
```

2.2.1 FDM for Poisson equation

Consider the following Poisson equations:

$$\begin{cases} -\Delta u = f & \text{in } \Omega = (x_a, x_b) \times (y_a, y_b), \\ u = 0 & \text{on } \partial\Omega. \end{cases}$$

With $h = \frac{x_b - x_a}{N+1}$ and $k = \frac{y_b - y_a}{N+1}$, we have the following approximation scheme by FDM:

$$-\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h^2} - \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{k^2} = f_{i,j},$$

for $i, j = 1, 2, \cdots, N$.

The five points stencil is given by

$$\begin{bmatrix} 0 & -\dfrac{1}{k^2} & 0 \\ -\dfrac{1}{h^2} & \dfrac{2}{h^2} + \dfrac{2}{k^2} & -\dfrac{1}{h^2} \\ 0 & -\dfrac{1}{k^2} & 0 \end{bmatrix}.$$

Let J be a tridiagonal matrix:

$$J = \operatorname{diag}(-1, 2, -1) := \begin{bmatrix} 2 & -1 & 0 & 0 & \cdots & 0 \\ -1 & 2 & -1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & -1 & 2 & -1 \\ 0 & \cdots & \cdots & 0 & -1 & 2 \end{bmatrix}$$

and let $J_x = (\frac{1}{h^2})\,J$ and $J_y = (\frac{1}{k^2})\,J$. Then we have the following matrix equation

$$UJ_x + J_yU = F.$$

From this equation, we have the following standard matrix form of Poisson Problem:

$$A * vecU = vecF$$

where

$$A = J_x \otimes I + I \otimes J_y.$$

```
-----------------------------------------------------------------------------------------

In Matlab
  >> h = (xb-xa)/(N+1);    k = (yb-ya)/(N+1);
  >> x = xa+h:h:xb-h;   y = ya+k:k:yb-k;
  >> [x,y] = meshgrid(x,y);       % matrices
  >> f = ft_f(x,y);   f = f(:);

  >> alp1 = -1/h^2; alp2 = 2/h^2; alp3 = -1/h^2;
  >> bet1 = -1/k^2; bet2 = 2/k^2; bet3 = -1/k^2;
  >> Jx = alp1*diag(ones(N-1,1),-1) + alp2*diag(ones(N,1)) + alp3*diag(ones(N-1,1),1);
  >> Jy = bet1*diag(ones(N-1,1),-1) + bet2*diag(ones(N,1)) + bet3*diag(ones(N-1,1),1);
  >> A = kron(Jx,eye(N)) + kron(eye(N),Jy);
  >> uh = A\f;

  >> Uh = reshape(uh, N, N);
  >> mesh(x, y, Uh)
-----------------------------------------------------------------------------------------
```

2.2.2 FDM for elliptic equation

Consider the following 2D elliptic problem:

$$\begin{cases} -\Delta u + \mathbf{b}\cdot\nabla u + cu &= f \qquad \text{in } \Omega = (0,1)^2, \\ u &= 0 \qquad \text{on } \partial\Omega, \end{cases} \tag{2D}$$

where $\mathbf{b} = (b^1, b^2)^t$ and c are constant functions.

With $h = k = 1/(N+1)$, we have the following approximate scheme:

$$\begin{aligned} &-\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h^2} - \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{k^2} \\ &+ b^1\frac{u_{i+1,j} - u_{i-1,j}}{2h} + b^2\frac{u_{i,j+1} - u_{i,j-1}}{2k} + c\,u_{i,j} = f_{i,j}, \end{aligned}$$

and then we have the following five points stencil

$$\begin{bmatrix} 0 & -\frac{1}{k^2} + \frac{b^2}{2k} & 0 \\ -\frac{1}{h^2} - \frac{b^1}{2h} & \frac{2}{h^2} + \frac{2}{k^2} + c & -\frac{1}{h^2} + \frac{b^1}{2h} \\ 0 & -\frac{1}{k^2} - \frac{b^2}{2k} & 0 \end{bmatrix}.$$

Let $\quad A = J_x \otimes I + I \otimes J_y$, where

$$J_x = \mathrm{diag}\Big(-\frac{1}{h^2} - \frac{b^1}{2h},\ \frac{2}{h^2} + c,\ -\frac{1}{h^2} + \frac{b^1}{2h}\Big) \quad \text{and} \quad J_y = \mathrm{diag}\Big(-\frac{1}{k^2} - \frac{b^2}{2k},\ \frac{2}{k^2},\ -\frac{1}{k^2} + \frac{b^2}{2k}\Big).$$

Then we have the matrix equation

$$UJ_x + J_yU = F \quad \text{or} \quad A * vecU = vecF.$$

You can also use the following tridiagonal matrices: with $A = J_x \otimes I + I \otimes J_y$, where

$$J_x = \mathrm{diag}\Big(-\frac{1}{h^2} - \frac{b^1}{2h},\ \frac{2}{h^2} + \frac{2}{k^2} + c,\ \ -\frac{1}{h^2} + \frac{b^1}{2h}\Big) \quad \text{and} \quad J_y = \mathrm{diag}\Big(-\frac{1}{k^2} - \frac{b^2}{2k},\ 0,\ \ -\frac{1}{k^2} + \frac{b^2}{2k}\Big).$$

```
-----------------------------------------------------------------------------------------

In Matlab
  >> N = input('The number of points used in each axis : N = ');
  >> h = 1/(N+1);
  >> x = h:h:N*h;   y = x;
  >> [x,y] = meshgrid(x,y);       % matrices
  >> f = ft_f(x,y);   f = f(:);

  >> alp1 = -1/h^2-b1/(2*h); alp2 = 2/h^2+c; alp3 = -1/h^2+b1/(2*h);
  >> bet1 = -1/h^2-b2/(2*h); bet2 = 2/h^2; bet3 = -1/h^2+b2/(2*h);
  >> Jx = alp1*diag(ones(N-1,1),-1) + alp2*diag(ones(N,1)) + alp3*diag(ones(N-1,1),1);
  >> Jy = bet1*diag(ones(N-1,1),-1) + bet2*diag(ones(N,1)) + bet3*diag(ones(N-1,1),1);

  >> A = kron(Jx,eye(N)) + kron(eye(N),Jy);
  >> uh = A\f;

  >> Uh = reshape(uh, N, N);
  >> mesh(x, y, Uh)


  You need to define the data function  'ft_f'.
---------------------------------------------------------------------------
```

Matlab Code : (FDM for Elliptic Problem 1)

```
% Filename fdm_ellip.m
% - Lapace(u) + b1*u_x + b2*u_y + c*u = f  in  O = (0,1)^2
%           u = 0  on boundary

function fdm_ellip(k,b1,b2,c)

if k>9, disp('Too big k !'); return; end
if nargin==1, c=0; b1=0; b2=0;
elseif nargin==2, b2=b1; c=0;
elseif nargin==3, c=0;
end

N = 2^k;  h = 1/(N+1);
x = h:h:N*h; y = x;     % [h 2h 3h ... Nh]
[x,y] = meshgrid(x,y);  % x, y are matrices.

% Right hand side ( Source term )
F = ft_f(x,y,b1,b2,c);  % matrix
F = F(:);       % column vector

% Coefficient matrix
alp1 = -1/h^2-b1/(2*h); alp3 = -1/h^2+b1/(2*h);
bet1 = -1/h^2-b2/(2*h); bet3 = -1/h^2+b2/(2*h);
alp2 = 4/h^2+c;
```

```
Jx = sparse(N,N); Jy = Jx; A = sparse(N^2,N^2);
Jx = alp1*diag(ones(N-1,1),-1) + alp2*diag(ones(N,1)) + alp3*diag(ones(N-1,1),1);
Jy = bet1*diag(ones(N-1,1),-1) + bet3*diag(ones(N-1,1),1);
A = kron(Jx, speye(N)) + kron(speye(N), Jy);
% Approximate solution
uh = A\F;       % vector
% Exact solution
u = ft_u(x,y);  % matrix
% vector -> matrix for mesh
uh = reshape(uh,N,N); % matrix
% Error
e = u-uh; err = h*norm(e(:));
fprintf('--- Elliptic problem with b = (%g,%g), c = %g --- \n',b1,b2,c);
fprintf('N = %g    l2-error = %12.4e \n',N,err);

subplot(131); meshc(x,y,u); title('Exact solution')
subplot(132); meshc(x,y,uh); title('Approximate solution')
subplot(133); meshc(x,y,e); title('Error plot')

%%%%%%%%%%%%%%%%%%%%%%%%%%
function u = ft_u(x,y)
  u = (x.^2-x).*sin(pi*y);  % (y.^2-y);

function f = ft_f(x,y,b1,b2,c)
  f = -2*sin(pi*y) + pi^2*(x.^2-x).*sin(pi*y);
  f = f + b1*(2*x-1).*sin(pi*y) + pi*b2*(x.^2-x).*cos(pi*y);
  f = f + c*ft_u(x,y);
```