In [1]:
```
%matplotlib inline
from math import *
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
```

Problem 1

(a)

In [2]:
```
# Polynomial
def poly(x, b):
    z = b[0]
    for i in range(1, n+1):
        z *= x
        z += b[i]
    return z

# degree of the interpolant polynomial
n = 4

# Data points
x = np.array([1, 2, 3, 4, 5])
y = np.array([1, 1, 2, 6, 24])

# Solve b using Vandermonde Matrix (monomial basis)
V = np.vander(x)
b = np.linalg.solve(V, y)
print('The polynomial\'s coefficients for x^4, x^3, x^2, x, and the intercept are: ')
print(b)
```

```
The polynomial's coefficients for x^4, x^3, x^2, x, and the intercept are: 
[  0.375       -3.41666667  11.625      -16.58333333   9.        ]
```

(b)

In [3]:
```
# Data points
x = np.array([1, 2, 3, 4, 5])
y = np.array([1, 1, 2, 6, 24])
y_log = np.log(np.array([1, 1, 2, 6, 24]))

# Solve b using monomial basis
V_log = np.vander(x)
b_log = np.linalg.solve(V, y_log)
print('The polynomial\'s(fitted on log(Gamma(n))) coefficients for x^4, x^3, x^2, x, and the intercept are: '
print(b_log)

```

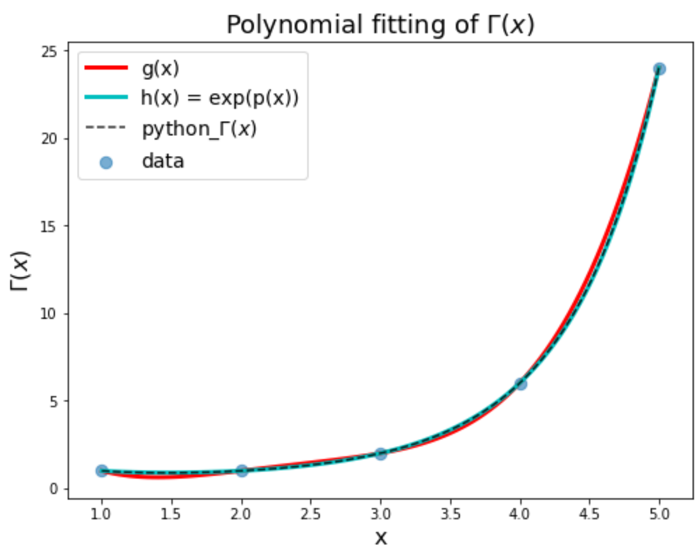
```
The polynomial's(fitted on log(Gamma(n))) coefficients for x^4, x^3, x^2, x, and the intercept are: 
[ 0.00707913 -0.11873828  0.88202509 -1.92109423  1.15072829]
```

(c)

In [4]:

```
from scipy.special import gamma

# Plot result
fig, ax = plt.subplots(1, 1, figsize=(8, 6))
X = np.linspace(1, 5, 100)
g = np.array([poly(XX, b) for XX in X])

p = np.array([poly(XX, b_log) for XX in X])
h = np.exp(p)
python_g = np.array([gamma(XX) for XX in X])

ax.plot(X, g, 'r-', linewidth=3, label='g(x)')
ax.plot(X, h, 'c-', linewidth=3,  label='h(x) = exp(p(x))')
ax.plot(X, python_g, 'k--', alpha=0.8, label='python_$\Gamma (x)$')

# Plot the raw data points
ax.scatter(x=x, y=y, marker='o', alpha=0.6, s=70, label='data')

ax.set_xlabel('x', fontsize=16)
ax.set_ylabel('$\Gamma (x)$', fontsize=16)
ax.set_title('Polynomial fitting of $\Gamma (x)$', fontsize=18)
ax.legend(fontsize=14)
```

Out[4]: <matplotlib.legend.Legend at 0x10d362f98>



(d)

In [5]:

```
# Compute maximum relative error of g(x) & h(x)
g_max_rel_e = np.max((np.abs(python_g - g)) / python_g)
g_max_rel_e_ind = np.argmax((np.abs(python_g - g)) / python_g)

h_max_rel_e = np.max((np.abs(python_g - h)) / python_g)
h_max_rel_e_ind = np.argmax((np.abs(python_g - h)) / python_g)

print('Max relative error between Gamma(x) and')
print('g(x) is {},    at x = {}'.format(g_max_rel_e, X[g_max_rel_e_ind]))
print('h(x) is {},    at x = {}'.format(h_max_rel_e, X[h_max_rel_e_ind]))
```

```
Max relative error between Gamma(x) and
g(x) is 0.2857257958185267,    at x = 1.404040404040404
h(x) is 0.011516203377661278,    at x = 1.3232323232323233
```

Problem 2

(a)

In [6]:

```python
# Arrange the coefficients of the cubics in the matrix form
A = np.zeros((9, 9))
A[0:6, 1:7] = 2*np.eye(6)
B = np.zeros((9, 9))
B[1:7, 0:6] = 2*np.eye(6)

M = 8*np.eye(9)
M[7, 7] = 0
M[8, 8] = 0
M[0, 8] = 2
M[7, 0] = -2
M[7, 8] = -4
M[8, 6] = 2
M[8, 7] = 4

M += (A+B)
M
```

Out[6]:

```
array([[ 8.,  2.,  0.,  0.,  0.,  0.,  0.,  0.,  2.],
       [ 2.,  8.,  2.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  2.,  8.,  2.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  2.,  8.,  2.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  2.,  8.,  2.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  2.,  8.,  2.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  2.,  8.,  0.,  0.],
       [-2.,  0.,  0.,  0.,  0.,  0.,  0.,  0., -4.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  2.,  4.,  0.]])
```

In [7]:

```python
# Solve the coefficients of the cubics for the 8 splines
q0 = np.array([-6, 0, 0, 0, 0, 0, 0, 6, 0])
q1 = np.array([-6, 0, 0, 0, 0, 0, 0, -6, 0])
q2 = np.array([6, 0, -6, 0, 0, 0, 0, 0, 0])
q3 = np.array([0, 6, 0, -6, 0, 0, 0, 0, 0])
q4 = np.array([0, 0, 6, 0, -6, 0, 0, 0, 0])
q5 = np.array([0, 0, 0, 6, 0, -6, 0, 0, 0])
q6 = np.array([0, 0, 0, 0, 6, 0, -6, 0, 0])
q7 = np.array([0, 0, 0, 0, 0, 6, 0, 0, -6])
q8 = np.array([0, 0, 0, 0, 0, 0, 6, 0, 6])

qs = [q0, q1, q2, q3, q4, q5, q6, q7, q8]
ps = []
for q in qs:
    p = np.linalg.solve(M, q)
    ps.append(p)
```

In [8]:

```
# The 4 basis cubics used in class note
def c0(x):
    return x*x*(3-2*x)
def c1(x):
    return -x*x*(1-x)
def c2(x):
    return (x-1)*(x-1)*x
def c3(x):
    return 2*x*x*x-3*x*x+1

# Define the 9 splines s0, s1, ..., s8, using coefficients solved above
def s0(x):
    params = ps[0]
    a,b,c,d,e,f,g,h,i = params[0],params[1],params[2],params[3],params[4],params[5],params[6],params[7],param
    if x>=0 and x<1: return c3(x) + a*c1(x) + i*c2(x)
    if x>=1 and x<2: return b*c1(x-1) + a*c2(x-1)
    if x>=2 and x<3: return c*c1(x-2) + d*c2(x-2)
    if x>=3 and x<4: return d*c1(x-3) + c*c2(x-3)
    if x>=4 and x<5: return e*c1(x-4) + d*c2(x-4)
    if x>=5 and x<6: return f*c1(x-5) + e*c2(x-5)
    if x>=6 and x<7: return g*c1(x-6) + f*c2(x-6)
    if x>=7 and x<=8: return h*c1(x-7) + g*c2(x-7)

def s1(x):
    params = ps[1]
    a,b,c,d,e,f,g,h,i = params[0],params[1],params[2],params[3],params[4],params[5],params[6],params[7],param
    if x>=0 and x<1: return c0(x) + a*c1(x) + i*c2(x)
    if x>=1 and x<2: return c3(x-1) + b*c1(x-1) + a*c2(x-1)
    if x>=2 and x<3: return c*c1(x-2) + d*c2(x-2)
    if x>=3 and x<4: return d*c1(x-3) + c*c2(x-3)
    if x>=4 and x<5: return e*c1(x-4) + d*c2(x-4)
    if x>=5 and x<6: return f*c1(x-5) + e*c2(x-5)
    if x>=6 and x<7: return g*c1(x-6) + f*c2(x-6)
    if x>=7 and x<=8: return h*c1(x-7) + g*c2(x-7)

def s2(x):
    params = ps[2]
    a,b,c,d,e,f,g,h,i = params[0],params[1],params[2],params[3],params[4],params[5],params[6],params[7],param
    if x>=0 and x<1: return a*c1(x) + i*c2(x)
    if x>=1 and x<2: return c0(x-1) + b*c1(x-1) + a*c2(x-1)
    if x>=2 and x<3: return c3(x-2) + c*c1(x-2) + d*c2(x-2)
    if x>=3 and x<4: return d*c1(x-3) + c*c2(x-3)
    if x>=4 and x<5: return e*c1(x-4) + d*c2(x-4)
    if x>=5 and x<6: return f*c1(x-5) + e*c2(x-5)
    if x>=6 and x<7: return g*c1(x-6) + f*c2(x-6)
    if x>=7 and x<=8: return h*c1(x-7) + g*c2(x-7)

def s3(x):
    params = ps[3]
    a,b,c,d,e,f,g,h,i = params[0],params[1],params[2],params[3],params[4],params[5],params[6],params[7],param
    if x>=0 and x<1: return a*c1(x) + i*c2(x)
    if x>=1 and x<2: return b*c1(x-1) + a*c2(x-1)
    if x>=2 and x<3: return c0(x-2) + c*c1(x-2) + d*c2(x-2)
    if x>=3 and x<4: return c3(x-3) + d*c1(x-3) + c*c2(x-3)
    if x>=4 and x<5: return e*c1(x-4) + d*c2(x-4)
    if x>=5 and x<6: return f*c1(x-5) + e*c2(x-5)
    if x>=6 and x<7: return g*c1(x-6) + f*c2(x-6)
    if x>=7 and x<=8: return h*c1(x-7) + g*c2(x-7)

def s4(x):
    params = ps[4]
    a,b,c,d,e,f,g,h,i = params[0],params[1],params[2],params[3],params[4],params[5],params[6],params[7],param
    if x>=0 and x<1: return a*c1(x) + i*c2(x)
    if x>=1 and x<2: return b*c1(x-1) + a*c2(x-1)
    if x>=2 and x<3: return c*c1(x-2) + d*c2(x-2)
    if x>=3 and x<4: return c0(x-3) + d*c1(x-3) + c*c2(x-3)
    if x>=4 and x<5: return c3(x-4) + e*c1(x-4) + d*c2(x-4)
    if x>=5 and x<6: return f*c1(x-5) + e*c2(x-5)
    if x>=6 and x<7: return g*c1(x-6) + f*c2(x-6)
    if x>=7 and x<=8: return h*c1(x-7) + g*c2(x-7)

def s5(x):
    params = ps[5]
    a,b,c,d,e,f,g,h,i = params[0],params[1],params[2],params[3],params[4],params[5],params[6],params[7],param
    if x>=0 and x<1: return a*c1(x) + i*c2(x)
    if x>=1 and x<2: return b*c1(x-1) + a*c2(x-1)
    if x>=2 and x<3: return c*c1(x-2) + d*c2(x-2)
    if x>=3 and x<4: return d*c1(x-3) + c*c2(x-3)
    if x>=4 and x<5: return c0(x-4) + e*c1(x-4) + d*c2(x-4)
    if x>=5 and x<6: return c3(x-5) + f*c1(x-5) + e*c2(x-5)
    if x>=6 and x<7: return g*c1(x-6) + f*c2(x-6)
```

```python
    if x>=7 and x<=8: return h*c1(x-7) + g*c2(x-7)

def s6(x):
    params = ps[6]
    a,b,c,d,e,f,g,h,i = params[0],params[1],params[2],params[3],params[4],params[5],params[6],params[7],param
    if x>=0 and x<1: return a*c1(x) + i*c2(x)
    if x>=1 and x<2: return b*c1(x-1) + a*c2(x-1)
    if x>=2 and x<3: return c*c1(x-2) + d*c2(x-2)
    if x>=3 and x<4: return d*c1(x-3) + c*c2(x-3)
    if x>=4 and x<5: return e*c1(x-4) + d*c2(x-4)
    if x>=5 and x<6: return c0(x-5) + f*c1(x-5) + e*c2(x-5)
    if x>=6 and x<7: return c3(x-6) + g*c1(x-6) + f*c2(x-6)
    if x>=7 and x<=8: return h*c1(x-7) + g*c2(x-7)

def s7(x):
    params = ps[7]
    a,b,c,d,e,f,g,h,i = params[0],params[1],params[2],params[3],params[4],params[5],params[6],params[7],param
    if x>=0 and x<1: return a*c1(x) + i*c2(x)
    if x>=1 and x<2: return b*c1(x-1) + a*c2(x-1)
    if x>=2 and x<3: return c*c1(x-2) + d*c2(x-2)
    if x>=3 and x<4: return d*c1(x-3) + c*c2(x-3)
    if x>=4 and x<5: return e*c1(x-4) + d*c2(x-4)
    if x>=5 and x<6: return f*c1(x-5) + e*c2(x-5)
    if x>=6 and x<7: return c0(x-6) + g*c1(x-6) + f*c2(x-6)
    if x>=7 and x<=8: return c3(x-7) + h*c1(x-7) + g*c2(x-7)

def s8(x):
    params = ps[8]
    a,b,c,d,e,f,g,h,i = params[0],params[1],params[2],params[3],params[4],params[5],params[6],params[7],param
    if x>=0 and x<1: return a*c1(x) + i*c2(x)
    if x>=1 and x<2: return b*c1(x-1) + a*c2(x-1)
    if x>=2 and x<3: return c*c1(x-2) + d*c2(x-2)
    if x>=3 and x<4: return d*c1(x-3) + c*c2(x-3)
    if x>=4 and x<5: return e*c1(x-4) + d*c2(x-4)
    if x>=5 and x<6: return f*c1(x-5) + e*c2(x-5)
    if x>=6 and x<7: return g*c1(x-6) + f*c2(x-6)
    if x>=7 and x<=8: return c0(x-7) + h*c1(x-7) + g*c2(x-7)

splines = [s0, s1, s2, s3, s4, s5, s6, s7, s8]

# Define the linear combinition of the splines (S)
def spline_comb(x, b):
    ret = 0
    for bi, si in zip(b, splines):
        ret += bi*si(x)
    return ret
```

In [9]:

```python
# Read in Xs, Ys from file
xs_1 = []
ys_1 = []
f1 = './midterm_files/sdata1.txt'
with open(f1) as f:
    lines = f.readlines()
    for l in lines:
        numbers = l.strip().split(' ')
        xs_1.append(float(numbers[0]))
        ys_1.append(float(numbers[1]))


# Arrange values of the 9 splines in matrix form
S = np.zeros((len(xs_1), 9))
for j, spline_k in enumerate(splines):
    for i, x in enumerate(xs_1):
        S[i, j] = spline_k(x)

# Linear least square fitting of the 9 splines on all data points (X, Y)
b1 = np.linalg.lstsq(S, np.array(ys_1))[0]
b1
```

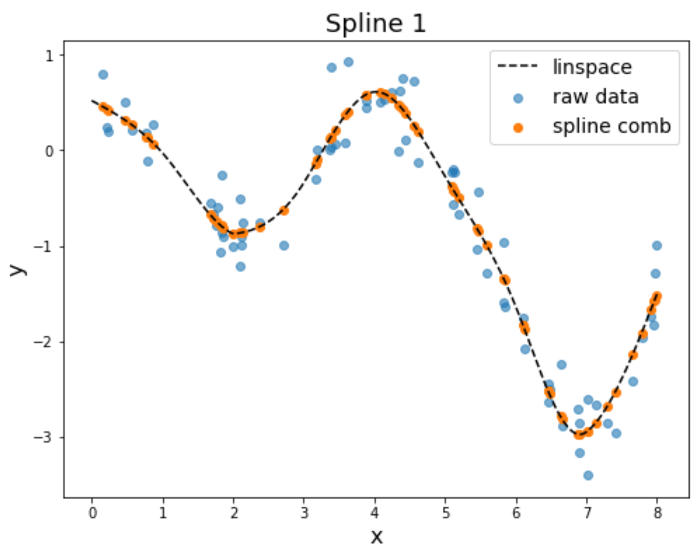
Out[9]:

```
array([ 0.51856191, -0.02703424, -0.87616979, -0.34414149,  0.60963002,
       -0.2735107 , -1.63220254, -2.94993774, -1.50771464])
```

In [10]:

```
y_spline_comb = np.array([spline_comb(x, b1) for x in xs_1])

x_lin = np.linspace(0, 8, 200)
y_spline_comb_lin = np.array([spline_comb(x, b1) for x in list(x_lin)]) # Linear spaced continuous points


# Plot results
fig, ax = plt.subplots(1, 1, figsize=(8, 6))
ax.scatter(x=xs_1, y=ys_1, alpha=0.6, label='raw data')
ax.scatter(x=xs_1, y=y_spline_comb, label='spline comb')
ax.plot(x_lin, y_spline_comb_lin, 'k--', label='linspace')
ax.set_xlabel('x', fontsize=16)
ax.set_ylabel('y', fontsize=16)
ax.set_title('Spline 1', fontsize=18)
ax.legend(fontsize=14)
```

Out[10]: <matplotlib.legend.Legend at 0x10d840518>



(b)

In [11]:

```
# Read in Xs, Ys from file
xs_2 = []
ys_2 = []
f2 = './midterm_files/sdata2.txt'
with open(f2) as f:
    lines = f.readlines()
    for l in lines:
        numbers = l.strip().split(' ')
        xs_2.append(float(numbers[0]))
        ys_2.append(float(numbers[1]))

# Arrange values of the 9 splines in matrix form
S2 = np.zeros((len(xs_2), 9))
for j, spline_k in enumerate(splines):
    for i, x in enumerate(xs_2):
        S2[i, j] = spline_k(x)

# Linear least square fitting of the 9 splines on all data points (X, Y)
b2 = np.linalg.lstsq(S, np.array(ys_2))[0]
b2
```

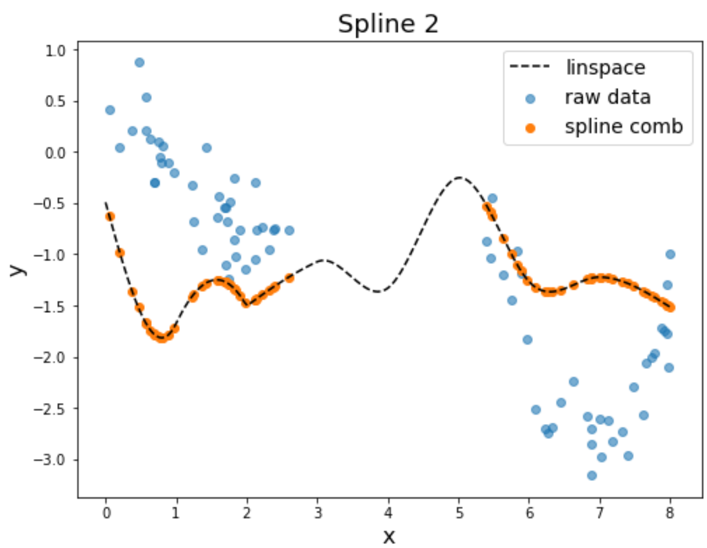
Out[11]:

```
array([-0.49281047, -1.67616518, -1.50517194, -1.07335582, -1.33135505,
       -0.25321862, -1.26349519, -1.22293233, -1.5130342 ])
```

In [12]:

```python
y_spline_comb_2 = np.array([spline_comb(x, b2) for x in xs_2])

x_lin = np.linspace(0, 8, 200)
y_spline_comb_lin = np.array([spline_comb(x, b2) for x in list(x_lin)]) # Linear spaced continuous points

# Plot results
fig, ax = plt.subplots(1, 1, figsize=(8, 6))
ax.scatter(x=xs_2, y=ys_2, alpha=0.6, label='raw data')
ax.scatter(x=xs_2, y=y_spline_comb_2, label='spline comb')
ax.plot(x_lin, y_spline_comb_lin, 'k--', label='linspace')
ax.set_xlabel('x', fontsize=16)
ax.set_ylabel('y', fontsize=16)
ax.set_title('Spline 2', fontsize=18)
ax.legend(fontsize=14)
```

Out[12]: <matplotlib.legend.Legend at 0x10da38e10>



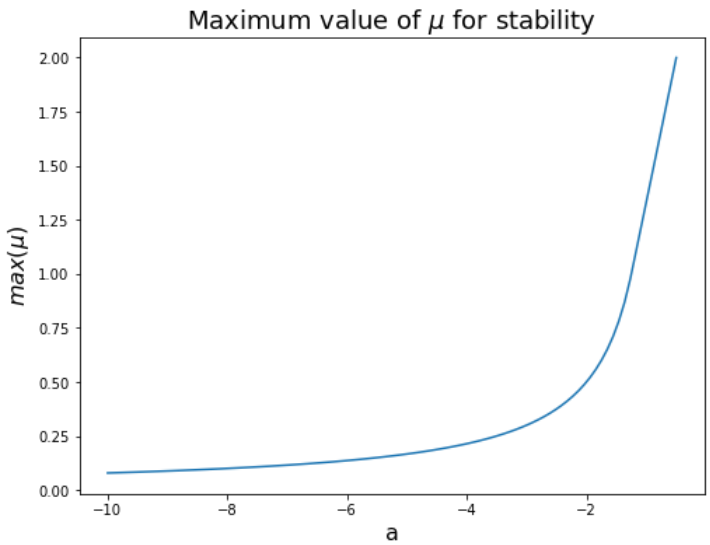
Problem 3

(f)

In [13]:

```python
# Maximum value of mu as a function of a
def mu(a):
    if a <= -5/4:
        return -3/(4*a+2)
    elif a <= -1/2:
        return (4*a+8)/3
    else:
        return 0

# Plot the maximum value of mu as a function of a
a_lin = np.linspace(-10, -0.5, 100)
mu_range = np.array([mu(a) for a in a_lin])

fig, ax = plt.subplots(1, 1, figsize=(8, 6))
ax.plot(a_lin, mu_range)
ax.set_xlabel('a', fontsize=16)
ax.set_ylabel('$max(\mu)$', fontsize=16)
ax.set_title('Maximum value of $\mu$ for stability', fontsize=18)
```

Out[13]: <matplotlib.text.Text at 0x10db99080>



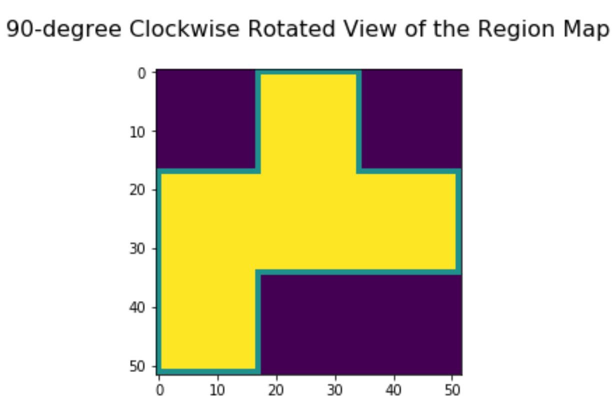
Problem 4

(c)

In [14]:

```python
# Grid setup
m = 52
mm = m*m
h = 3.0/(m-1)
beta = 1.0/(5*h*h)
alpha = -8.0*beta
s = 1344

type_map = np.ones((m, m))

# mark exterior points
for j in range(0, 17):
    for k in range(0, 17):
        type_map[j, k] = -1
for j in range(0, 17):
    for k in range(35, 52):
        type_map[j, k] = -1
for j in range(35, 52):
    for k in range(18, 52):
        type_map[j, k] = -1

# mark boundary
for k in range(1, 18):
    type_map[17, k] = 0
for j in range(0, 17):
    type_map[j, 17] = 0
for k in range(18, 35):
    type_map[0, k] = 0
for j in range(1, 18):
    type_map[j, 34] = 0
for k in range(35, 52):
    type_map[17, k] = 0
for j in range(18, 35):
    type_map[j, 51] = 0
for k in range(17, 51):
    type_map[34, k] = 0
for j in range(35, 52):
    type_map[j, 17] = 0
for k in range(0, 17):
    type_map[51, k] = 0
for j in range(17, 51):
    type_map[j, 0] = 0

plt.title('90-degree Clockwise Rotated View of the Region Map\n', fontsize=16)
plt.imshow(type_map)
```

Out[14]: <matplotlib.image.AxesImage at 0x10d3eaf28>



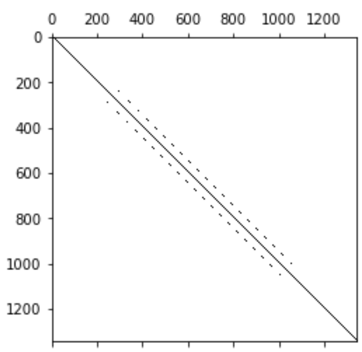
In [15]:

```python
reindex_dic = {} # mapping 2D index to 1D
rev_reindex_dic = {} # mapping 1D index back to 2D
count = 0
for j in range(m):
    for k in range(m):
        if type_map[j, k] == 1:
            reindex_dic[(j, k)] = count
            rev_reindex_dic[count] = (j, k)
            count += 1

print('The number of interior points in this region is {}'.format(count))
```

The number of interior points in this region is 1344

In [16]:

```
# Create derivative matrix and source term
A = np.zeros((s, s))
F = np.zeros((s))

ind = 0 # track the 1D indexing of interior points
for j in range(m):
    x = j*h
    for k in range(m):
        if type_map[j, k] != 1:
            continue

        A[ind, reindex_dic[(j, k)]] = 1 - alpha

        orth_neighbors = np.array([(j-2, k), (j-1, k), (j+1, k), (j+2, k),
                                   (j, k-2), (j, k-1), (j, k+1), (j, k+2)])
        for jn, kn in orth_neighbors:
            if jn<0 or jn>=m or kn<0 or kn>=m : # neighbor is exterior: apply ghost node
                A[ind, reindex_dic[(j, k)]] += beta
            elif type_map[jn, kn] == 1: # neighbor is interior
                A[ind, reindex_dic[(jn, kn)]] = -beta
            elif type_map[jn, kn] == -1: # neighbor is exterior: apply ghost node
                A[ind, reindex_dic[(j, k)]] += beta

        # Source term
        y = k*h
        F[ind] = 3 - (x-1.5)**2 + (y-1.5)**2

        ind += 1

plt.spy(A)
plt.show()
```



In [17]:

```
# ||A - A.T||_{Frob.} == 0
np.linalg.norm(A - A.T, ord='fro')
```

Out[17]: 0.0

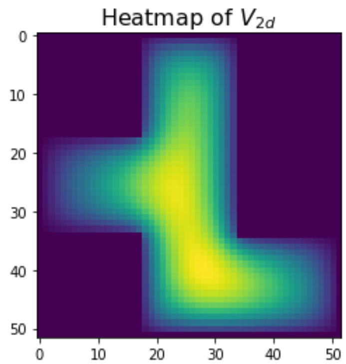
In [18]:

```
# Cholesky factorize A = L*L^T
L = np.linalg.cholesky(A)

# Solve for B: L*B = F
B = np.linalg.solve(L, F)

# Solve for V: L.T*V = B
V = np.linalg.solve(L.T, B)
```

In [19]:

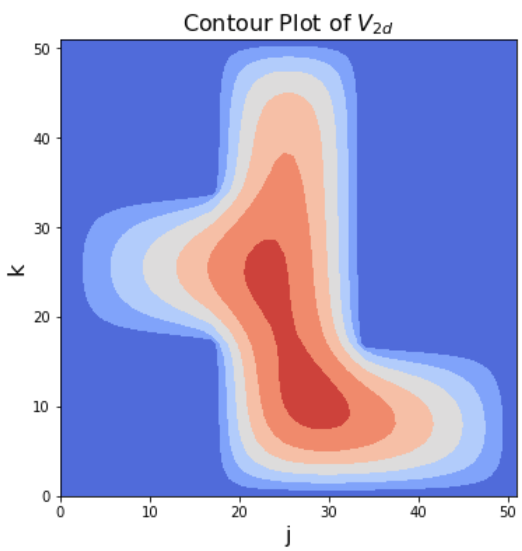
```python
# Recover 1D indexing to 2D
V_2d = np.zeros((m, m))
for i in range(s):
    j, k = rev_reindex_dic[i]
    V_2d[m-1-k, j] = V[i] # Correct array indexing to match (x, y) coordinates

plt.title('Heatmap of $V_{2d}$', fontsize=16)
plt.imshow(V_2d)
```

Out[19]: `<matplotlib.image.AxesImage at 0x10da8d9e8>`



In [20]:

```python
# Plot the solution for V on the (x, y) plane
fig, ax = plt.subplots(1, 1, figsize=(8, 6))
ax.contourf(np.arange(m), np.arange(m)[::-1], V_2d, cmap=plt.cm.coolwarm)

ax.set_xlabel('j', fontsize=16)
ax.set_ylabel('k', fontsize=16)
plt.title('Contour Plot of $V_{2d}$', fontsize=16)
plt.gca().set_aspect('equal')
```



(d)

In [21]:

```python
for i in range(s):
    if V[i] == np.max(V):
        j_max, k_max = rev_reindex_dic[i]
        print('Maximun of V = {}'.format(V[i]))
        print('At j = {}, k = {}'.format(j_max, k_max))
```

```
Maximun of V = 0.5147104874225262
At j = 28, k = 11
```

In [ ]: