Fill in group number and member names:

In [1]:
```
GROUP = "3"
NAME1 = "Nils Romanus Myrberg"
NAME2 = "Måns Karp"
```

Optimization for learning - FRTN50

Assignment 1

The goal of this assignment is to become familiar with some of the steps involved in solving an optimization problem. In this assignment, you will form Fenchel-dual problems, find gradients and/or proximal operators, and implement the proximal gradient method.

Problem The problem we will solve is the following constrained problem

$$\underset{x \in S}{\text{minimize}} \ \frac{1}{2} x^T Q x + q^T x \tag{1}$$

where $Q \in \mathbb{S}^n_{++}$, $q \in \mathbb{R}^n$ and $S \subseteq \mathbb{R}^n$ is a set defined by the points $a, b \in \mathbb{R}^n$, $a \leq b$, such that

$$S = \{x \in \mathbb{R}^n : a \leq x \leq b\}.$$

I.e., we are going to minimize a quadratic function over an n-dimensional box. Recall that, the vector inequality $a \leq b$ means that

$$a_i \leq b_i$$

for each $i = 1, \ldots, n$. Define the function $f : \mathbb{R}^n \to \mathbb{R}$ such that

$$f(x) = \frac{1}{2} x^T Q x + q^T x$$

for each $x \in \mathbb{R}^n$ and let $\iota_S : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ denote the indicator function of the set S, i.e.

$$\iota_S(x) = \begin{cases} 0 & \text{if } x \in S, \\ \infty & \text{if } x \in \mathbb{R}^n \setminus S. \end{cases}$$

Problem (1) can then be written as

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \ f(x) + \iota_S(x). \tag{2}$$

Solution method To solve optimization problem (2), we will use the *proximal gradient method*. It solves problems of the form

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \ f(x) + g(x) \tag{3}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is differentiable and $g : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ is proximable, i.e., $\text{prox}_{\gamma g}$ can be cheaply computed. The proximal gradient method (with constant step-size) is given by:

- Pick some arbitrary initial guess $x^0 \in \mathbb{R}^n$ and step-size $\gamma > 0$.
- For $k = 0, 1, 2 \ldots$, let \begin{align}\label{eq:pg}\tag{4}
  x^{k+1} = \prox_{\gamma g}\left(x^k - \gamma \nabla f(x^k)\right).
  \end{align}
- Stop when x^k is deemed to have converged.

In this assignment, we simply run the proximal gradient method a large fixed number of iterations and plot the norm of the step-length/residual, $\|x^{k+1} - x^k\|_2$, of each step to make sure it converges to zero. Since the experiments are run on a computer, zero means smaller than machine precision, which usually is around 10^{-15}.

The step-size parameter γ in the $(???)$ will affect the convergence. It should be tuned to the problem or chosen based on properties of f and g. In particular, suppose that f and g are proper, closed and convex. If f is β-smooth for some parameter $\beta > 0$, the maximal step-size to guarantee convergence is $\gamma < \frac{2}{\beta}$.

Below are the tasks that you need to solve. Keep this in mind:

- The suggested exercises in the exercise compendium found on the Canvas course page, up until and including the chapter "Proximal gradient method - basics", is relevant for this assignment.
- Carefully motivate every step in your calculations.
- Use figures and tables to motivate your answers.
- Figures must have appropriately labeled axes and must be referenced in the main text.
- Your code should be written in a quite general manner, i.e., if a question is slightly modified, it should only require slight modifications in your code as well.
- Comment your code well.
- Make sure you plot in such a way that small quantities (e.g., $\|x^{k+1} - x^k\|_2$) are visible. In particular, use log-linear plots, where the quantity that should go to 0 is on the y-axis using logarithmic scale, and the iteration number k on the x-axis using linear scale.
- What you need to submit to Canvas:
  - This jupyter notebook containing your solutions.
  - An exported pdf version of the jupyter notebook.

Task 1:

Show that f and ι_S in (2) are convex and show that constraint qualification (CQ) holds. You are allowed to assume that $\text{relint } S \neq \emptyset$. Note that f and ι_S also are closed, but you do not need to prove this.

Solution:

The Hessian of f is well defined and $\nabla^2 f(x) = Q$. Since $Q \in \mathbb{S}^n_{++}$, convexity of f follows from the second order condition for convexity.

Since ι_S is a constant function (hence convex) on its effective domain we need only show that S is a convex set to show that ι_S is a convex function. This is shown in expression (1), where $x, y \in S$ and $\theta \in [0, 1]$.

$$a \leq \theta x + (1-\theta)y \leq \theta b + (1-\theta)b \leq b \implies \theta x + (1-\theta)y \in S \tag{1}$$

Constraint qualification holds since

$$\operatorname{relint}(\operatorname{dom} \iota_S) = \operatorname{relint} S \neq \emptyset \tag{1}$$

$$\operatorname{relint}(\operatorname{dom} f) = \mathbb{R}^n \tag{2}$$

$$\implies \operatorname{relint}(\operatorname{dom} \iota_S) \cap \operatorname{relint}(\operatorname{dom} f) \neq \emptyset. \tag{3}$$

Task 2:

Compute the conjugate functions f^* and ι_S^*. The final expressions are not allowed to be given implicitly via optimization problems. E.g., projection formulas must be solved explicitly.

Solution:

f^*, The Fenchel conjugate of f is computed by using the Fenchel-Young equality displayed in equation (4).

$$f(x^*) = s^T x^* - f^*(s) \quad iff \quad s \in \partial f(x^*) \tag{4}$$

Where $\partial f(x^*)$ denotes the subdifferential of f at x^*. Since f is differentiable on $\mathbb{R}^n$ its subdifferential is the singleton set displayed in expression (5).

$$\partial f(x) = \{\nabla f(x)\} = \{Qx + q\} \quad \forall \quad x \in \mathbb{R}^n \tag{5}$$

Using the singleton set in expression 5 in the condition for the Fenchel-Young equality one arrives at expression 6.

$$s = Qx^* + q \Leftrightarrow x^* = Q^{-1}(s - q) \tag{6}$$

Where Q^{-1} is well-defined since $Q \in \mathbb{S}^n_{++}$. Using expression 6 in the Fenchel-Young equality 4 one arrives at expression 7

$$f^*(s) = (s^T x^* - f(x^*) = \frac{1}{2}(Q^{-1}(s-q))^T Q(Q^{-1}(s-q)) - q^T(Q^{-1}(s-q))) =$$

ι_S^*, The Fenchel conjugate of ι_S is computed by using the conjugate definition. One may note that ι_S is a separable function that can be written on the form displayed in expression 8

$$\iota_S(x) = \sum_{i=1}^{n} \iota_{[a_i,b_i]}(x_i) \tag{8}$$

where a_i, b_i and x_i denotes component with index $i \in \{1, 2...., n\}$ of each vector a, b, x respectively. Using the definition of the conjugate one may compute the conjugate of the scalar component $\iota_{[a_i,b_i]}(x_i)$ of $\iota_S(x)$ with index $i \in \{1, 2...., n\}$ as shown in expression 9.

$$\iota^*_{[a_i,b_i]}(s_i) = \sup_{x_i \in \mathbb{R}} (s_i x_i - \iota_{[a_i,b_i]}(x_i)) = \sup_{x_i \in [a_i,b_i]} (s_i x_i) = \begin{cases} s_i a_i & s_i < 0 \\ 0 & s_i = 0 \\ s_i b_i & s_i > 0 \end{cases} \tag{9}$$

Since $\iota_S(x)$ is a separable, proper, closed and convex function its Fenchel conjugate is also separable, resulting in expression 10.

$$\iota^*_S(s) = \sum_{i=1}^{n} \iota^*_{[a_i,b_i]}(s_i) \Leftrightarrow \begin{cases} v_i = \begin{cases} s_i a_i & s_i < 0 \\ 0 & s_i = 0 \\ s_i b_i & s_i > 0 \end{cases} \\ \iota^*_S(s) = \sum_{i=1}^{n} v_i \end{cases} \tag{10}$$

Task 3:

Write down a Fenchel-dual problem to (2). Show that constraint qualification for the dual problem (CQ-D) holds.

Attention/hint: Keep track of your minus signs.

Solution:

A Fenchel dual problem is given by

$$\underset{\mu \in \mathbb{R}^n}{\text{minimize}} f^*(-\mu) + \iota^*_S(\mu). \tag{11}$$

This is dervied by the following procedure. Let x^* solve the following problem

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} f(x) + \iota_S(x). \tag{12}$$

Since f and ι_S are closed convex functions and constraint qualification holds, this is equivalent to

$$0 \in \partial f(x^*) + \partial \iota_S(x^*) \tag{13}$$

Now let $\mu^* \in \partial \iota_S(x^*)$, then the above is in turn equivalent to

$$\begin{cases} \mu^* \in \partial \iota_S(x^*) \\ -\mu^* \in \partial f(x^*). \end{cases} \tag{14}$$

Since f and ι_S are closed convex functions, this is equivalent to

$$\begin{cases} x^* \in \partial \iota_S^*(\mu^*) \\ x^* \in \partial f^*(-\mu^*). \end{cases} \tag{15}$$

Which in turn is equivalent to

$$0 \in \partial \iota_S^*(\mu^*) - \partial f^*(-\mu^*). \tag{16}$$

This is equivalent to that μ^* solves the following problem

$$\underset{\mu \in \mathbb{R}^n}{\text{minimize}} f^*(-\mu) + \iota_S^*(\mu). \tag{17}$$

Constraint qualification holds for this problem since

$$\operatorname{relint}(\operatorname{dom} \iota_S^*) = \operatorname{relint}(\operatorname{dom} f^*) = \mathbb{R}^n \tag{18}$$

$$\implies \operatorname{relint}(\operatorname{dom} \iota_S^*) \cap \operatorname{relint}(\operatorname{dom} f^*) \neq \emptyset. \tag{19}$$

Task 4:

Show that f and f^* are β-, and β^*-smooth, respectively. Find expressions for the smallest such parameters β and β^*.

Hint: Later when calculating the smoothness parameters in Pyhton, make sure to read the documentation carefully so that you use the correct function.

Solution:

A function f is β-smooth if its gradient is β-Lipschitz continous, i.e if its gradient fulfills the condition displayed in equation 20.

$$\|\nabla f(x) - \nabla f(y)\|_2 \leq \beta \|x - y\|_2 \quad \forall \quad x, y \in \mathbb{R}^n \tag{20}$$

Using the definition coupled with the definition of an operator norm, and letting f be the aforementioned quadratic function, one arrives at expression 21.

$$\|\nabla f(x) - \nabla f(y)\|_2 = \|Qx - q - (Qy - q)\|_2 = \|Q(x - y)\|_2 \leq \|Q\|_2 \|x - y\|_2 =$$

Where $\|Q\|_2$ denotes the spectral norm of Q which is its largest eigenvalue denoted $\lambda_{\max}(Q)$ which is always positive since $Q \in \mathbb{S}^n_{++}$. Comparing expression 21 with the definition in equation 20 one may identify $\beta = \|Q\|_2$. Performing the same operation for the the Fenchel-conjugate of f one arrives at expression 22.

$$\|\nabla f^*(x) - \nabla f^*(y)\|_2 = \|Q^{-1}(x - q) - Q^{-1}(y - q)\|_2 = \|Q^{-1}(x - y)\|_2 \leq \|Q^-$$

Once again comparing expression 22 with the defintion in equation 20 one may identify $\beta^* = \|Q^{-1}\|_2$ which is, once again, well-defined since $Q \in \mathbb{S}^n_{++}$.

Task 5:

Compute ∇f, ∇f^*, $\text{prox}_{\gamma\iota_S}$ and $\text{prox}_{\gamma\iota_S^*}$. The final expressions are not allowed to be given implicitly via optimization problems. E.g., projection formulas must be solved explicitly.

Solution:

As shown in task 2 $\nabla f(x) = Qx + q$. $\nabla f^*(s) : \mathbb{R}^n \mapsto \mathbb{R}$ is computed by the steps listed in expression 23

$$\nabla f^*(s) = \nabla(\frac{1}{2}(s-q)^T Q^{-1}(s-q)) = Q^{-1}(s-q) \tag{23}$$

$\text{prox}_{\gamma\iota_S}$ is computed by noting that the ι_S is seperable and using the fact that expression 24 holds for the separable ι_S.

$$\iota_S(x) = \sum_{i=1}^{n} \iota_{[a_i,b_i]}(x_i) \Leftrightarrow \text{prox}_{\gamma\iota_S}(z) = \begin{bmatrix} \text{prox}_{\gamma\iota_{[a_1,b_1]}}(z_1) \\ \vdots \\ \text{prox}_{\gamma\iota_{[a_n,b_n]}}(z_n) \end{bmatrix} \tag{24}$$

The proximal operator can subsequently be computed by considering the scalar case for an arbitrary index $i \in \{1, \dots, n\}$ and using the definition of the proximal operator as shown in expression 25.

$$\text{prox}_{\gamma\iota_{[a_i,b_i]}}(z_i) = \underset{x\in\mathbb{R}}{\text{argmin}}(\iota_{[a_i,b_i]}(z_i) + \frac{1}{2\gamma}\|x - z_i\|_2^2) = \underset{x\in[a_i,b_i]}{\text{argmin}}(\frac{1}{2\gamma}\|x - z_i\|_2^2) = \Pi$$

Which can be rewritten, using a more compact representation, as expression 26.

$$\begin{cases} v_i = \begin{cases} a_i & v_i \leq a_i \\ v_i & v_i \in (a_i, b_i) \\ b_i & v_i \geq b_i \end{cases} \\ \text{prox}_{\gamma\iota_S}(z) = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} \end{cases} \tag{26}$$

Since ι_S is convex one may use Moreau decomposition to compute the proximal operator of its Fenchel conjugate as shown in expression 27.

$$\text{prox}_{\gamma\iota_S^*}(z) = z - \gamma\,\text{prox}_{\gamma^{-1}\iota_S}(\gamma^{-1}z) \Leftrightarrow \begin{cases} \nu_i = \begin{cases} z_i - \gamma a_i & z_i \leq a_i\gamma \\ 0 & z_i \in (a_i\gamma, b_i\gamma) \\ z_i - \gamma b_i & z_i \geq b_i\gamma \end{cases} \\ \text{prox}_{\gamma\iota_S^*}(z) = \begin{bmatrix} \nu_1 \\ \vdots \\ \nu_n \end{bmatrix} \end{cases} \tag{27}$$

Task 6:

Based on your results above, write explicitly out the proximal gradient update rule (4) for both the primal and the dual problem. Use x as the primal variable and μ as the dual variable.

Attention/hint: Keep track for your minus signs.

Solution:

Consider the composite optimization problem displayed in expression 28.

$$\underset{x\in\mathbb{R}^n}{\text{minimize}}\, f(x) + g(x)) \tag{28}$$

If one assumes f to be β-smooth and g closed convex one may replace f with its coinciding quadratic upper bound and minimize in every iteration in order to arrive at the proximal gradient update scheme displayed in 29.

$$x^{k+1} = \text{prox}_{\gamma,g}(x^k - \gamma^k\nabla f(x^k)) \tag{29}$$

Where x^k denotes x at iteration k. As shown in tasks 4 and 1, the quadratic function $f(x) = \frac{1}{2}x^TQx + q^Tx$ is β-smooth, and $g(x) = \iota_S(x)$ closed convex. The proximal gradient update shceme for the primal problem displayed in 29 hence becomes expression 30.

$$x^{k+1} = \text{prox}_{\gamma,\iota_S}(x^k - gamma^k(Qx^k + q)) \tag{30}$$

Using the result from task 5 one may explicitly write the primal proximal update scheme as expression 31

$$x^{k+1} = \text{prox}_{\gamma,\iota_S}(x^k - \gamma^k(Qx^k + q)) = \begin{bmatrix} e_1^k \\ \vdots \\ e_n^k \end{bmatrix} \qquad e_i^k = \begin{cases} a_i & if \quad x_i^k + \gamma^k(\sum_{j=} \\ x_i^k + \gamma^k(\sum_{j=1}^n Q_{ij}x_i^k + \\ b_i & if \quad x_i^k + \gamma^k(\sum_{j=} \end{cases}$$

Where $(\cdot)_i^k$ denotes component with index i of a vector at iteration k. Note that expression 31 can be written in the more compact way displayed in expression 32.

$$\begin{cases} v^k = x^k - \gamma^k(Qx^k + q) \\ x_i^{k+1} = \begin{cases} a_i & v_i^k \leq a_i \\ v_i^k & v_i^k \in (a_i, b_i) \\ b_i & v_i^k \geq b_i \end{cases} \end{cases} \tag{32}$$

Consider the dual problem derived in task 5 displayed in equation 33.

$$\underset{\mu \in \mathbb{R}^n}{\text{minimize}} f^*(-\mu) + \iota_S^*(\mu) \tag{33}$$

In the dual problem case f^* is β-smooth, due to strong convexity of f, and ι_S^* closed convex yielding a proximal update scheme displayed in equation 34.

$$\mu^{k+1} = \text{prox}_{\gamma_k, \iota_S^*}(\mu^k + \gamma^k \nabla f^*(-\mu^k)) = \text{prox}_{\gamma_k, \iota_S^*}(\mu^k + \gamma^k(Q^{-1}(-\mu^k - q))) \tag{34}$$

Since $\text{prox}_{\gamma_k, \iota_S^*}$ is known from task 5 the proximal update update scheme for the dual problem can be explicetely written as expression 35 using the compact representation.

$$\begin{cases} \nu^k = \mu^k + \gamma^k(Q^{-1}(-\mu^k - q)) \\ \mu_i^{k+1} = \begin{cases} \nu_i^k - \gamma^k a_i & \nu_i^k \leq a_i \gamma^k \\ 0 & \nu_i^k \in (\gamma^k a_i, \gamma^k b_i) \\ \nu_i^k - \gamma^k b_i & \nu_i^k \geq b_i \gamma^k \end{cases} \end{cases} \tag{35}$$

Task 7:

Suppose that $\mu^\star \in \mathbb{R}^n$ is an optimal solution to the dual problem you found in Task 3. Given $\mu^\star$, and starting from the optimality condition for the dual problem (given by Fermat's rule), recover an optimal point $x^\star \in \mathbb{R}^n$ to the primal problem (2), and show that this $x^\star$ is in fact an optimal solution to the primal problem (2).

Solution:

Given $\mu^* \in \mathbb{R}^n$ is an optimal solution to the dual problem, expression 36 holds by Fermat's rule.

$$0 \in \partial(\iota_S^*(\mu^*) + f^*(-\mu^*)) = \partial \iota_S^*(\mu^*) - \partial f^*(-\mu^*) \tag{36}$$

Where the subdifferential of the sum is the sum of subdifferentials due to convexity of ι_S^* and ∂f^* and the fact that the CQ holds. By letting $x^* \in \partial f^*(-\mu^*)$ equation 36 is equivalent to expression 37.

$$\begin{cases} x^* \in \partial f^*(-\mu^*) \\ x^* \in \partial \iota_S^*(\mu^*) \end{cases} \tag{37}$$

Expression 37 is equivalent to the dual displayed in expression 38 since $\partial f = (\partial f^*)^{-1}$ if f is a proper closed convex function.

$$\begin{cases} -\mu^* \in \partial f(x^*) \\ \mu^* \in \partial \iota_S(x^*) \end{cases} \tag{38}$$

Since f^* is differentiable for all $\mu \in \mathbb{R}$ its subdifferential is the singleton set $\partial f^*(-\mu^*) = \{\nabla f^*(-\mu^*)\}$. Therefore equation 37 becomes expression 39.

$$\begin{cases} x^* \in \partial f^*(-\mu^*) = \{\nabla f^*(-\mu^*)\} \\ x^* \in \partial \iota_S^*(\mu^*) \end{cases} \Leftrightarrow x^* = \nabla f^*(-\mu^*) \Leftrightarrow x^* = Q^{-1}(-\mu^* - q)$$

i.e the first row of equation 37 uniquely characterizes x^* resulting in the optimality condition, for the primal problem, beeing fulfilled as displayed in expression 40.

$$0 \in \partial(f(x^*) + \iota_s(x^*)) = \{\nabla f(x^*)\} + \partial \iota_s(x^*) = \{Q(Q^{-1}(\mu^* - q)) + q) + \mu^*\} =$$

which is equivalent to $x^* = \nabla f^*(-\mu^*) = Q^{-1}(-\mu^* - q)$ minimizes the primal problem.

Task 8:

Use your results above to fill in the functions below.

Solution:

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm
%matplotlib inline

def quad(x,Q,q):
    """
    quad(x,Q,q) computes the quadratic function (1/2)x'Qx + q'x

    :param x: the variable of the quadratic function
    :param Q: the matrix in the quadratic function that corresponds to the q
    :param q: the vector in the quadratic function that corresponds to the l
    :return: (1/2)x'Qx + q'x
    """

    return x.T @ Q @ x + q.T @ x

def quadconj(mu,Q,q):
    """
    quadconj(mu,Q,q) computes the conjugate function of the
    quadratic function (1/2)x'Qx + q'x, evaluated at mu

    :param mu: the variable of the conjugate function
    :param Q: the matrix in the quadratic function that corresponds to the q
    :param q: the vector in the quadratic function that corresponds to the l
    :return: conjugate of (1/2)x'Qx + q'x, evaluated at mu
    """
    # Write your solution here
    return 1/2 * (mu - q).T @ np.linalg.inv(Q) @ (mu - q)
```

```
def box(x,a,b):
    """
    box(x,a,b) computes the indicator function of the box contraint
    [a,b]

    :param x: the variable of the indicator function
    :param a: the left vector defining the box contraint
    :param b: the right vector defining the box contraint
    :return: 0 if x is in [a,b] and infinity otherwise
    """
    if np.all(a <= x) and np.all(x <= b):
        return 0
    else:
        return np.Inf

def boxconj(mu,a,b):
    """
    boxconj(mu,a,b) computes the conjugate function of the indicator functic
    of the box contraint [a,b], evaluated at mu

    :param mu: the variable of the conjugate function
    :param a: the left vector defining the box contraint
    :param b: the right vector defining the box contraint
    :return: conjugate of the indicator function of the box contraint [a,b],
    """
    # Write your solution here
    return sum(np.array([m * a if m <= 0 else 0  if m == 0 else m * b for m

def grad_quad(x,Q,q):
    """
    grad_quad(x,Q,q) computes the gradient of the quadratic function (1/2)x'

    :param x: the variable of the quadratic function
    :param Q: the matrix in the quadratic function that corresponds to the q
    :param q: the vector in the quadratic function that corresponds to the l
    :return: gradient of (1/2)x'Qx + q'x
    """
    # Write your solution here
    return Q @ x + q

def grad_quadconj(mu,Q,q):
    """
    grad_quadconj(mu,Q,q) computes the gradient of the conjugate function of
    the quadratic function (1/2)x'Qx + q'x, evaluated at mu

    :param mu: the variable of the conjugate function
    :param Q: the matrix in the quadratic function that corresponds to the q
    :param q: the vector in the quadratic function that corresponds to the l
    :return: gradient of the conjugate of (1/2)x'Qx + q'x, evaluated at mu
    """
    # Write your solution here
    return np.linalg.inv(Q) @ (mu - q)

def prox_box(x,a,b,gamma):
    """
    prox_box(x,a,b,gamma) computes proximal operator of the indicator functi
    of the box contraint [a,b], evaluated at x

    :param x: the variable of the poximal operator
    :param a: the left vector defining the box contraint
    :param b: the right vector defining the box contraint
    :param gamma: the step-size parameter
    :return: proximal operator of the indicator function of the
    box contraint [a,b], evaluated at x
```

```
    """
    # Write your solution here
    return np.array([a_ if x_ <= a_
                     else x_ if (x_ >= a_ and x_ <= b_)
                     else b_ for x_,a_,b_ in zip(x,a,b)])

def prox_boxconj(mu,a,b,gamma):
    """
    prox_box(mu,a,b,gamma) computes proximal operator of the conjugate funct
    the indicator function of the box contraint [a,b], evaluated at mu

    :param mu: the variable of the poximal operator
    :param a: the left vector defining the box contraint
    :param b: the right vector defining the box contraint
    :param gamma: the step-size parameter
    :return: proximal operator of the conjugate function of the indicator fu
    box contraint [a,b], evaluated at mu
    """
    # Write your solution here
    return np.array([mu_ - gamma * a_ if mu_ <= gamma * a_
                     else 0 if (mu_ >= a_ * gamma and mu_ <= b_ * gamma)
                     else mu_ - gamma * b_ for mu_,a_,b_ in zip(mu,a,b)])

def dual_to_primal(mu,Q,q,a,b):
    """
    dual_to_primal(mu,Q,q,a,b) computes the solution x* to the primal proble
    given a solution mu* to the dual problem.

    :param mu: the dual variable
    :param Q: the matrix in the quadratic function that corresponds to the q
    :param q: the vector in the quadratic function that corresponds to the l
    :param a: the left vector defining the box contraint
    :param b: the right vector defining the box contraint
    :return: the extracted primal variable
    """
    # Write your solution here
    return grad_quadconj(-mu,Q,q)
```

Task 9:

Below is a function for generating Q, q, a, and b that define the quadratic function f and the box constraint set S. Use Task 8 to solve the primal problem using the proximal gradient method.

a) What seems to be the best choice of γ?

b) Does the upper bound $\gamma < \frac{2}{\beta}$ seem reasonable?

Test different initial points for the algorithm:

c) Does this affect the point the algorithm converges to?

d) Reason about why/why not it affects the final point. *Hint:* Look at the objective function in (2).

e) Does your final point x^{final} satisfy the constraint $x^{\text{final}} \in S$?

f) What about the iterates, do they always satisfy the constraint, $x^k \in S$? Why/why not?

Solution:

a)

$\frac{2}{\beta}$ is the theoretically best step size since it is the largest whilst still guaranteeing convergence. After some experimentation one may however note that the algorithm still converges if $\beta = \frac{2.044}{\beta}$ is used. One can probably find an optimal γ that is greater than $\frac{2}{\beta}$ by doing more experiments but from our testing it seems that approximately $\frac{2}{\beta}$ is the best choice.

b)

The upper bound seems reasonable as the solution starts to diverge around $\frac{2}{\beta}$ (see image 2 below).

c)

No, after some testing the algorithm result seems to be the same for different initializations.

d)

This makes sense as the objective function in (2) is a sum of a strongly convex function and a convex function which implies that the objective function $f(x) + \iota_s(x)$ is a strongly convex function. A strongly convex function has a unique minimizer and hence the algorithm will converge to this unique minimizer if it converges, which it will do if the step size is appropriate as shown above. We know that $f(x)$ is strongly convex by the second order condition for strong convexity for twice continously differentiable functions as $\nabla^2 f(x) = Q \succ 0$ and Q is a constant matrix implying that $\nabla^2 f(x) = Q \succ \sigma I$.

e)

Yes the final iterate satisfies the constraint as shown below.

f)

The only iterate that can not satisfy the condition is the first one if we start outside the box. Otherwise every iterate satisfy the condition.

In [3]:
```
def problem_data():
    """
    problem_data() generates the problem data variables Q, q, a and b

    :return: (Q,q,a,b)
    """
    rs = np.random.RandomState(np.random.MT19937(np.random.SeedSequence(1)))
    n = 20
    Q = rs.randn(n,n)
    Q = Q.T@Q
    q = rs.randn(n)
```

```
    a = -rs.rand(n)
    b = rs.rand(n)
    return Q, q, a, b

(Q,q,a,b) = problem_data()

# Write your solution here
```

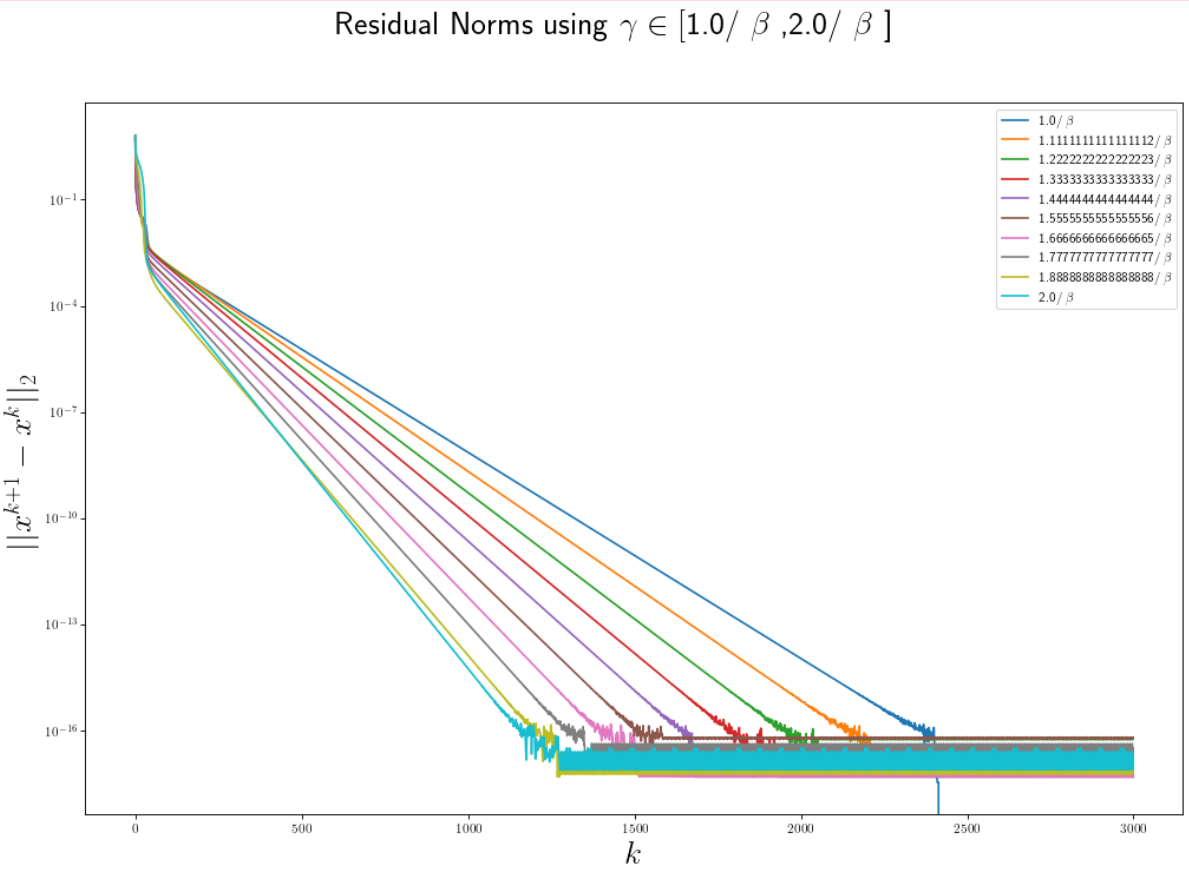
In [4]:

```
from matplotlib.pyplot import figure
plt.rcParams['text.usetex'] = True

def get_gammas(Q, start = 1, stop = 2, num = 10, is_dual = False):
    beta = np.linalg.norm(Q, ord = 2)
    if is_dual:
        gammas_labels = [str(i) + r'/ $ \beta^{\ast} $ ' for i in np.linspac
    else:
        gammas_labels = [str(i) + r'/ $ \beta $ ' for i in np.linspace(start
    gammas = [i/beta for i in np.linspace(start,stop,num)]
    return gammas, gammas_labels

def solve_primal(gamma, num_iters = 1e3):
    n = 20
    rs = np.random.RandomState(np.random.MT19937(np.random.SeedSequence(1)))
    x = rs.randn(n)
    fs = []
    xs = []
    diffs = []
    iteration = []
    for i in range(0, int(num_iters)):
        xs.append(x)
        temp = x
        f = quad(x,Q,q)
        x = prox_box(x - gamma * grad_quad(x,Q,q),a,b,gamma)
        fs.append(f)
        diffs.append(np.linalg.norm(x-temp, ord = 2))
    message = 'Final iterate is in S' if (np.all(a <= xs[-1]) and np.all(b >
    else 'Final iterate is not in S'
    print(message)
    return [diffs, fs, xs]

def get_primal_sol_curves(gammas, num_iters = 1e3):
    return [solve_primal(gamma, num_iters)[0] for gamma in gammas]

def plot_sol_curves(diffs, gamma_labels = None,  gammas = None, is_dual = Fa
    fig = figure(figsize=(15, 10), dpi=80)
    if is_dual:
        title = r'Residual Norms using $\gamma^{\ast} \in [$' + gamma_labels
        ylab = r'$||\mu^{k+1} - \mu^{k}||_2$'
    else:
        title = r'Residual Norms using $\gamma \in [$' + gamma_labels[0] + '
        ylab = r'$||x^{k+1} - x^{k}||_2$'
    if gamma_labels and gammas:
        fig.suptitle(title, fontsize=25)
    for gamma, gamma_label, diff in tqdm(zip(gammas, gamma_labels, diffs)):
        if gamma_labels is not None and gammas is not None:
            plt.semilogy(diff, label = gamma_label)
        else:
            plt.semilogy(diff)
    plt.legend()
    y_lab = plt.ylabel(ylab, fontsize=25)
    x_lab = plt.xlabel(r'$k$', fontsize=25)
    plt.show()
```

In [5]:
```
gammas, gamma_labels = get_gammas(Q, start = 1, stop = 2, num = 10)
diffs = get_primal_sol_curves(gammas, num_iters = 3e3)
plot_sol_curves(diffs, gamma_labels = gamma_labels,  gammas = gamma_labels)
```

```
Final iterate is in S
Final iterate is in S
Final iterate is in S
Final iterate is in S
Final iterate is in S
Final iterate is in S
Final iterate is in S
Final iterate is in S
Final iterate is in S
Final iterate is in S
```

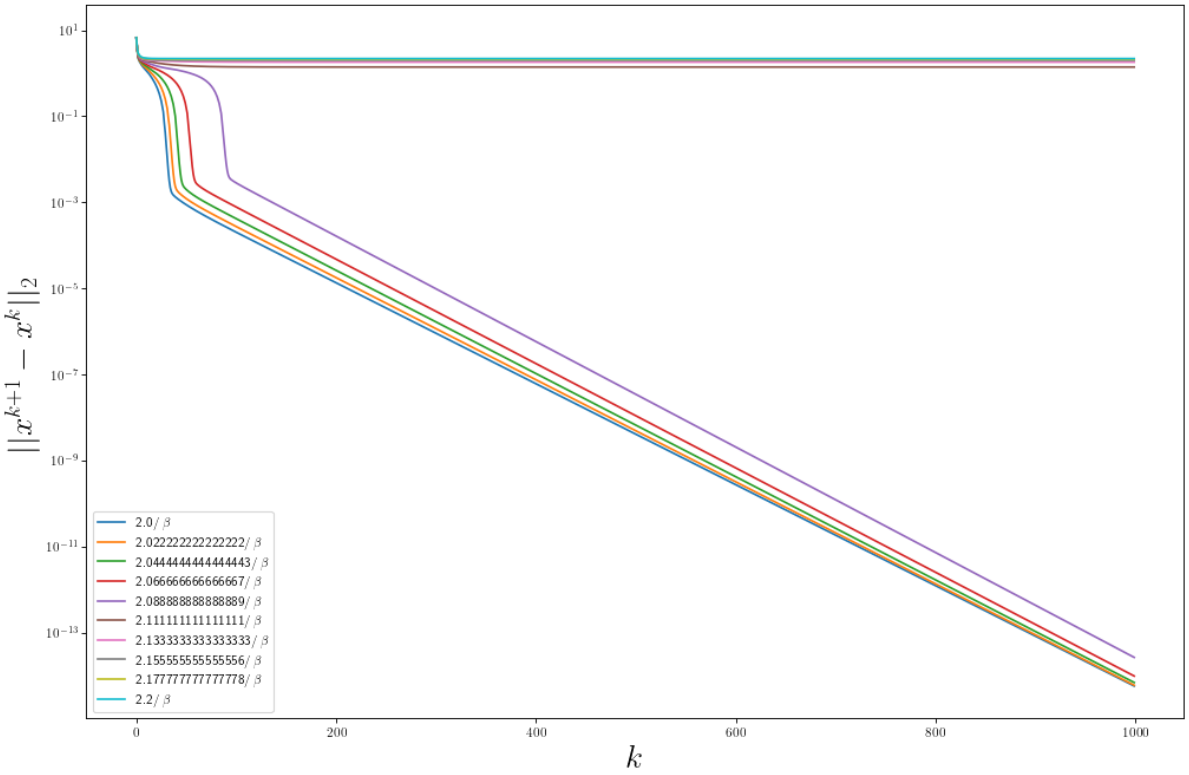
```
10it [00:00, 48.58it/s]
```



In [6]:
```
gammas, gamma_labels = get_gammas(Q, start = 2, stop = 2.2, num = 10)
diffs = get_primal_sol_curves(gammas, num_iters = 1e3)
plot_sol_curves(diffs, gamma_labels = gamma_labels,  gammas = gammas)
```

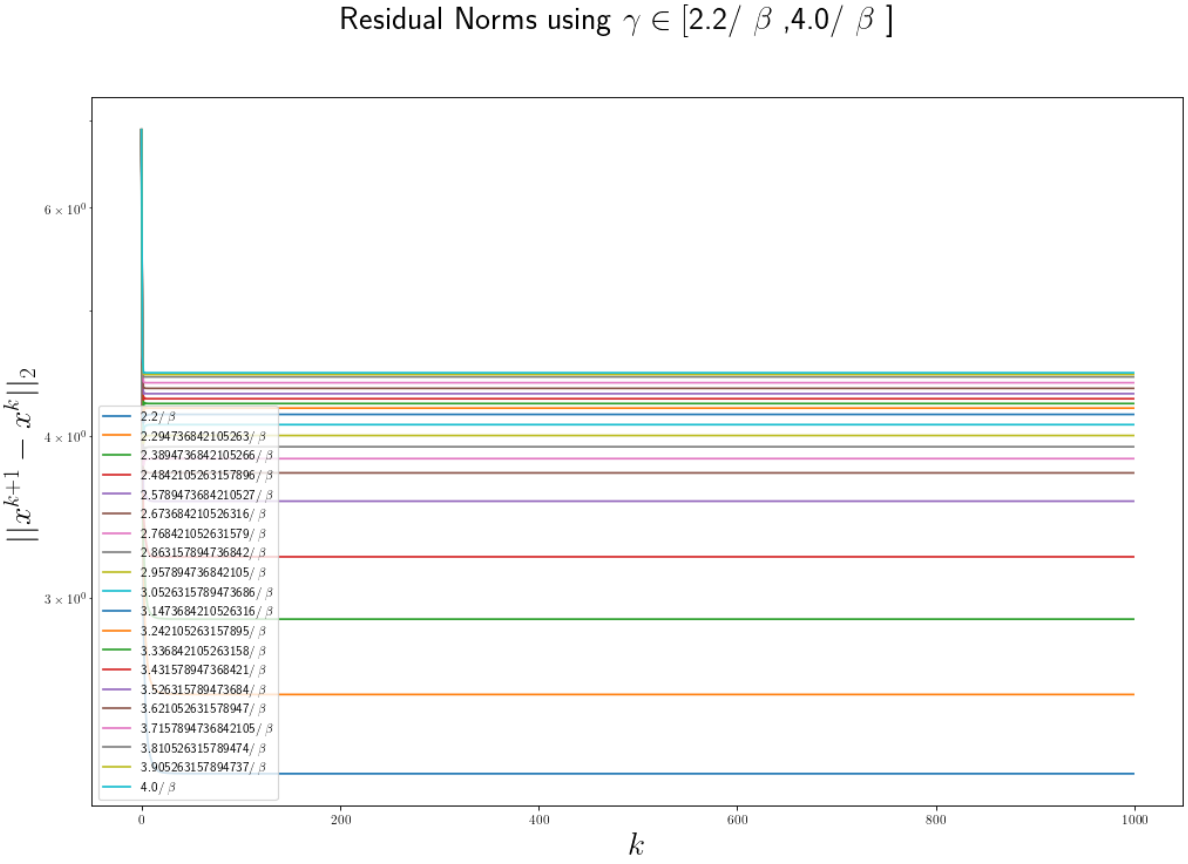
```
Final iterate is in S
Final iterate is in S
Final iterate is in S
Final iterate is in S
Final iterate is in S
Final iterate is in S
Final iterate is in S
Final iterate is in S
Final iterate is in S
Final iterate is in S
```

```
10it [00:00, 119.91it/s]
```



In [7]:
```
gammas, gamma_labels = get_gammas(Q, start = 2.2, stop = 4, num = 20)
diffs = get_primal_sol_curves(gammas, num_iters = 1e3)
plot_sol_curves(diffs, gamma_labels = gamma_labels,  gammas = gammas)
```

```
Final iterate is in S
Final iterate is in S
Final iterate is in S
Final iterate is in S
Final iterate is in S
Final iterate is in S
Final iterate is in S
Final iterate is in S
Final iterate is in S
Final iterate is in S
Final iterate is in S
Final iterate is in S
Final iterate is in S
Final iterate is in S
Final iterate is in S
Final iterate is in S
Final iterate is in S
Final iterate is in S
Final iterate is in S
Final iterate is in S
20it [00:00, 142.06it/s]
```



Task 10:

Solve the dual problem.

a) Similar to the previous task, find/verify the upper bound on the step-size and find a good step-size choice.

Let x^{final} be the final points from Task 9 and μ^{final} the final point for the dual problem. Let $\hat{x}^{\text{final}}$ final primal points extracted from the final dual point μ^{final} using the expression from Task 7:

b) Are x^{final} and $\hat{x}^{\text{final}}$ the same?

c) Is $\hat{x}^{\text{final}}$ in the box S?

d) Let μ^k be the iterates of the dual method, using the expression from Task 7, extract the primal iterates $\hat{x}^k$ from μ^k. Does $\hat{x}^k$ always satisfy the constraint $\hat{x}^k \in S$?

Also:

e) How do the function values $f\left(\hat{x}^k\right)$ develop over the iterations?

f) What about $f\left(\hat{x}^k\right) + \iota_S\left(\hat{x}^k\right)$?

Solution:

a)

$\frac{2}{\beta^*}$ is the theoretically best step size since it is the largest whilst still guaranteeing convergence. After some experimentation one may however note that the algorithm still converges if one uses $\gamma^* = \frac{2.055}{\beta^*}$.

b)

Yes, the results from task 7 matches reality, as shown in the end of the notebook where the euclidean norm of the difference, between the final iterates retrieved from both the dual- and primal solutions, is very small.

c)

No.

d)

No.

e)

The function values of the iterates $\{f(\hat{x}_k)\}_{k\in\{1,\dots,N\}}$ is an oscillating series that goes to the minimal value as k increases as shown below.

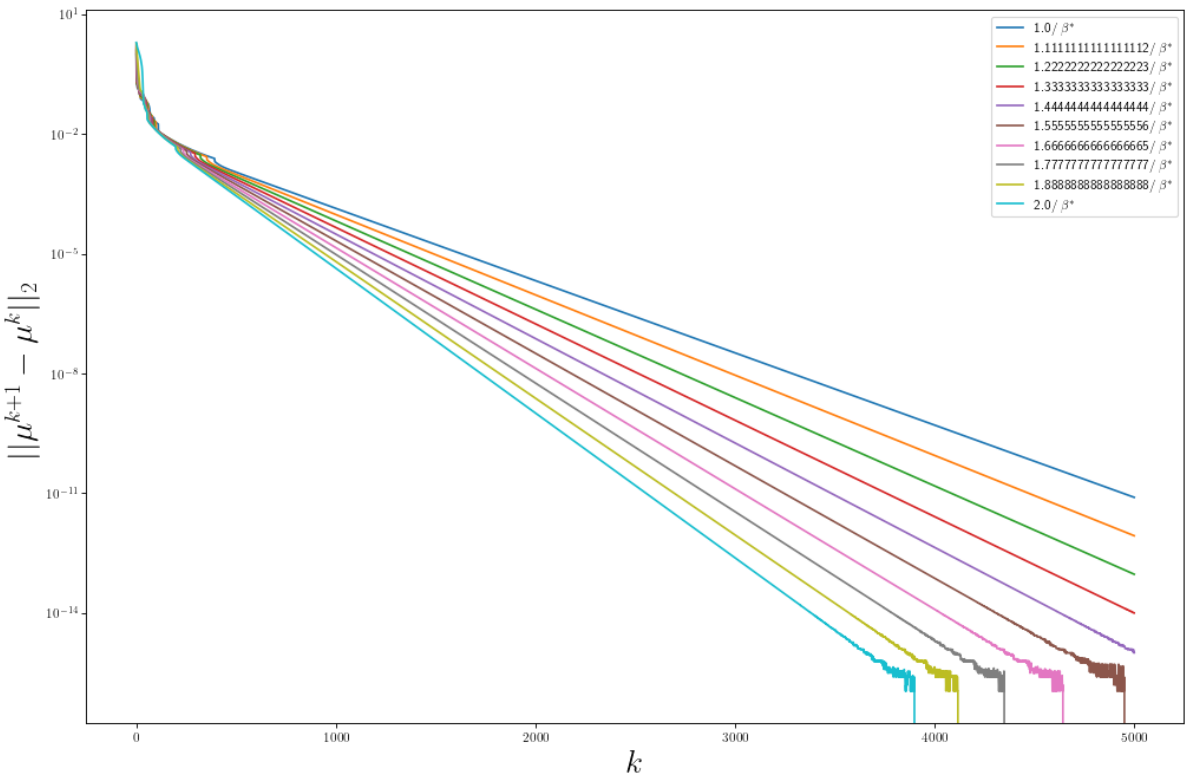
f)

The function values of the iterates $\{f(\hat{x}_k) + \iota_S(\hat{x}_k)\}_{k\in\{1,\dots,N\}}$ is infinity everywhere since none of the recovered iterates are in the box as shown below.

In [8]:
```
def solve_dual(gamma, num_iters = 1e3):
    n = 20
    rs = np.random.RandomState(np.random.MT19937(np.random.SeedSequence(1)))
    x = rs.randn(n)
    fs = []
    xs = []
    diffs = []
    iteration = []
    for i in range(0, int(num_iters)):
        xs.append(x)
        temp = x
        f = quadconj(-x,Q,q)
        x = prox_boxconj(x + gamma * grad_quadconj(-x,Q,q),a,b,gamma)
        fs.append(f)
        diffs.append(np.linalg.norm(x-temp, ord = 2))
    return [diffs, fs, xs]

def get_dual_sol_curves(gammas, num_iters = 1e3):
    return [solve_dual(gamma, num_iters)[0] for gamma in gammas]
```

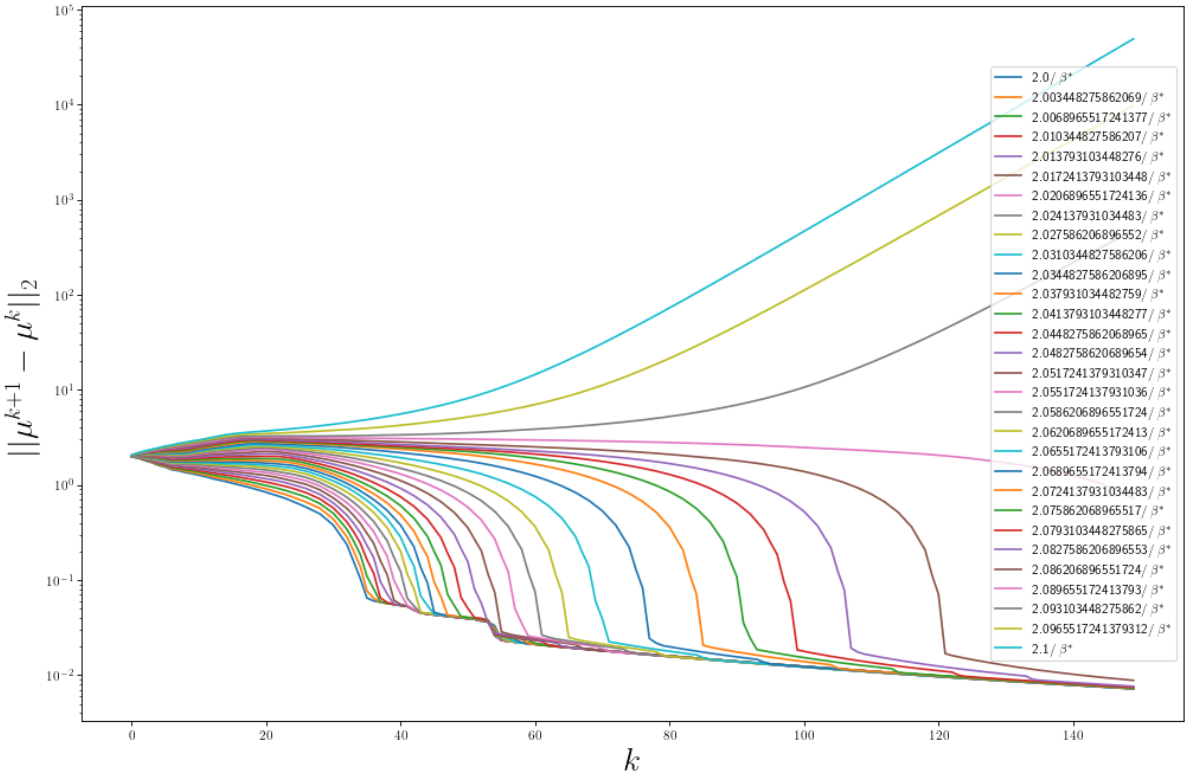
In [9]:
```
gammas, gamma_labels = get_gammas(np.linalg.inv(Q), start = 1, stop = 2, num
diffs = get_dual_sol_curves(gammas, num_iters = 5e3)
plot_sol_curves(diffs, gamma_labels = gamma_labels,  gammas = gammas, is_dua
```

```
10it [00:00, 149.75it/s]
```



In [10]:
```
gammas, gamma_labels = get_gammas(np.linalg.inv(Q), start = 2, stop = 2.1, n
diffs = get_dual_sol_curves(gammas, num_iters = 1.5e2)
plot_sol_curves(diffs, gamma_labels = gamma_labels,  gammas = gammas, is_dua
```

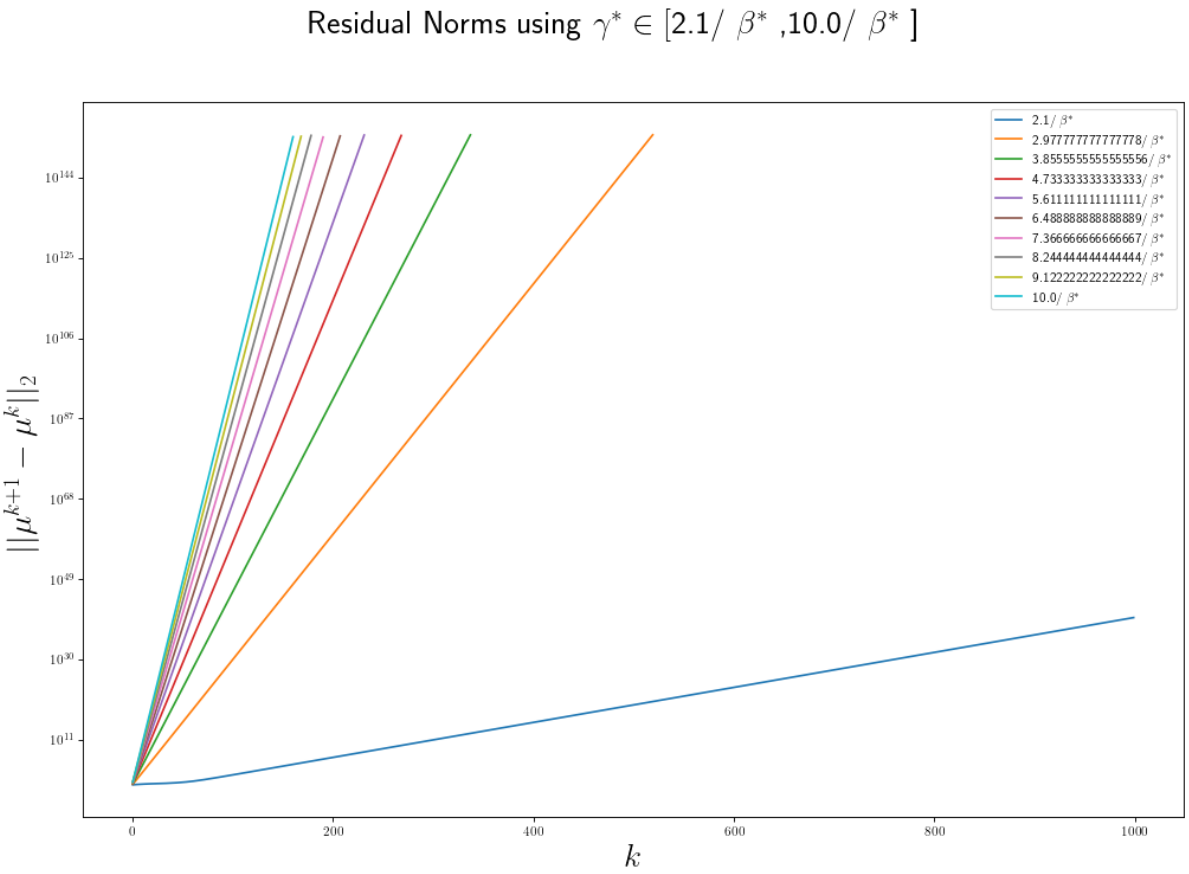
```
30it [00:00, 368.27it/s]
```



In [11]:
```
gammas, gamma_labels = get_gammas(np.linalg.inv(Q), start = 2.1, stop = 10,
diffs = get_dual_sol_curves(gammas, num_iters = 1e3)
```

```
plot_sol_curves(diffs, gamma_labels = gamma_labels,  gammas = gammas, is_dua
```

```
/var/folders/bq/myp7pv7d2kx50tbcnmg3pjgm0000gn/T/ipykernel_18245/1769790779.
py:29: RuntimeWarning: overflow encountered in matmul
  return 1/2 * (mu - q).T @ np.linalg.inv(Q) @ (mu - q)
/var/folders/bq/myp7pv7d2kx50tbcnmg3pjgm0000gn/T/ipykernel_18245/1769790779.
py:82: RuntimeWarning: overflow encountered in matmul
  return np.linalg.inv(Q) @ (mu - q)
/var/folders/bq/myp7pv7d2kx50tbcnmg3pjgm0000gn/T/ipykernel_18245/1153368219.
py:13: RuntimeWarning: invalid value encountered in add
  x = prox_boxconj(x + gamma * grad_quadconj(-x,Q,q),a,b,gamma)
/var/folders/bq/myp7pv7d2kx50tbcnmg3pjgm0000gn/T/ipykernel_18245/1769790779.
py:82: RuntimeWarning: invalid value encountered in matmul
  return np.linalg.inv(Q) @ (mu - q)
/var/folders/bq/myp7pv7d2kx50tbcnmg3pjgm0000gn/T/ipykernel_18245/1769790779.
py:29: RuntimeWarning: invalid value encountered in matmul
  return 1/2 * (mu - q).T @ np.linalg.inv(Q) @ (mu - q)
10it [00:00, 154.41it/s]
```



Showing that primal recovery yields same solution

In [12]:

```
_, _, xs_primal = solve_primal(2/np.linalg.norm(Q, ord = 2), num_iters = 1e4
_, _, mus_dual = solve_dual(2/np.linalg.norm(np.linalg.inv(Q), ord = 2), num
x_dual = dual_to_primal(mus_dual[-1],Q,q,a,b) #call primal to dual on final
x_primal = xs_primal[-1]
primal_dual_diff = np.linalg.norm(x_primal - x_dual, ord = 2) #Compare
message = 'Final iterate recovered from dual is in S' if (np.all(a <= x_dual
            else 'Final iterate recovered from dual is not in S'
print(message)
```

```
Final iterate is in S
Final iterate recovered from dual is not in S
```

In [13]:
```
print(f"Difference between dual and primal final iterate is: {primal_dual_di
```

```
Difference between dual and primal final iterate is: 4.152181583040242e-15
```

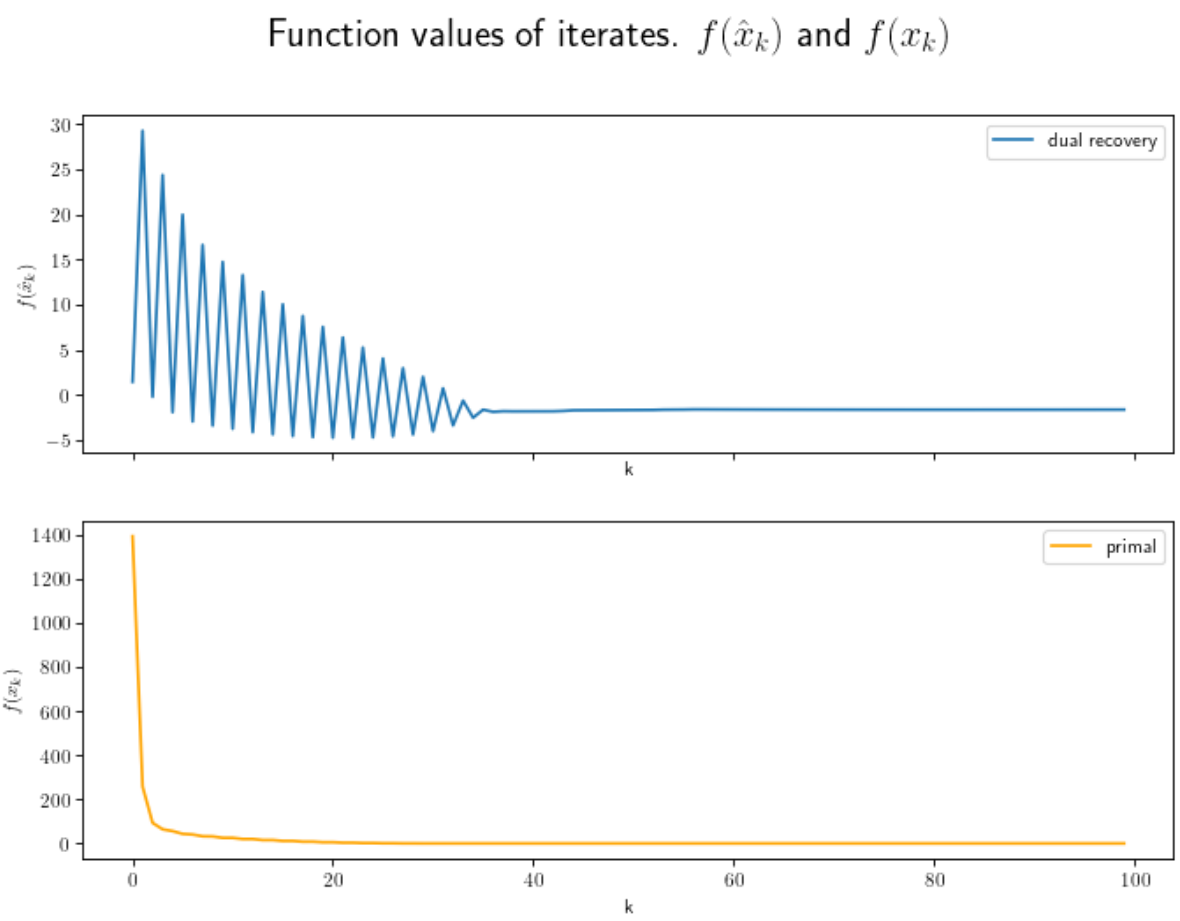
In [14]:
```
xs_recovered = [dual_to_primal(m,Q,q,a,b) for m in mus_dual]
in_s = 0
not_in_s = 0
for x_r in xs_recovered:
    if (np.all(a <= x_r) and np.all(b >= x_r)):
        in_s += 1

    else:
        not_in_s += 1
print(f'Iterates in s: {in_s} \nIterates not in s: {not_in_s}')
```

```
Iterates in s: 0
Iterates not in s: 10000
```

Investigating Function Values

In [15]:
```
%matplotlib inline
fig, ax = plt.subplots(2, figsize=(10, 7), dpi=80, sharex = True)
ax[0].plot([quad(x_r,Q,q) for x_r in xs_recovered[0:100]], label = "dual rec
ax[1].plot([quad(x_r,Q,q) for x_r in xs_primal[0:100]], label = 'primal', co
plt.xlabel('$k$')
ax[0].legend()
ax[0].set(xlabel='k', ylabel= r'$f(\hat{x}_k)$')
ax[1].set(xlabel='k', ylabel= r'$f(x_k)$')
ax[1].legend()
fig.suptitle("Function values of iterates. $f(\hat{x}_k)$ and $f(x_k)$", fon
```

Out[15]:
```
Text(0.5, 0.98, 'Function values of iterates. $f(\\hat{x}_k)$ and $f(x_k)$')
```



In [16]:
```
fig, ax = plt.subplots(2, figsize=(10, 7), dpi=80, sharex = True)
ax[0].plot([quad(x_r,Q,q) + box(x_r,a,b) for x_r in xs_recovered[0:100]], la
ax[1].plot([quad(x_r,Q,q) + box(x_r,a,b) for x_r in xs_primal[0:100]], label
plt.xlabel('$k$')
ax[0].legend()
ax[0].set(xlabel='k', ylabel= r'$f(\hat{x}_k)$')
ax[1].set(xlabel='k', ylabel= r'$f(x_k)$')
ax[1].legend()
fig.suptitle("Function values of iterates. $f(\hat{x}_k)$ and $f(x_k)$", fon
```

Out[16]:
```
Text(0.5, 0.98, 'Function values of iterates. $f(\\hat{x}_k)$ and $f(x_k)$')
```